Science of Computer Programming 160 (2018) 3–29

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Choreography Realizability Enforcement through the Automatic Synthesis of Distributed Coordination Delegates

Marco Autili *, Paola Inverardi, Massimo Tivoli

Università dell'Aquila, Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica, L'Aquila, Italy

ARTICLE INFO

Article history:
Received 31 March 2016
Received in revised form 6 October 2017
Accepted 22 October 2017
Available online 27 October 2017

Keywords:
Choreography realizability
Realizability enforcement
Software integration
Automated synthesis
Service choreography
Distributed coordination

ABSTRACT

In the near future we will be surrounded by a virtually infinite number of software applications that provide services in the digital space. This situation promotes reuse-based software production through composition of existing software services distributed over the Internet. Choreographies are a form of decentralized composition that model the external interaction of the participant services by specifying peer-to-peer message exchanges from a global perspective. When third-party (possibly black-box) services are to be composed, obtaining the distributed *coordination* logic required to enforce the realizability of the specified choreography is a non-trivial and error prone task. Automatic support is then needed. In this paper, we propose a formal approach to the enforcement of choreography realizability through the automatic synthesis of distributed *Coordination Delegates* (CDs). CDs are additional software entities with respect to the choreography participants. They are synthesized in order to proxify and control the participant services interaction. When interposed among the services, the synthesized entities enforce the collaboration prescribed by the choreography specification. We state correctness of our synthesis method and illustrate its formalization at work on an explanatory example.

1. Introduction

The Future Internet [1] promotes a distributed-computing environment that will be increasingly inhabited by a virtually infinite number of software services. Software systems will be more and more built by composing together software services distributed over the Internet. This calls for new networking paradigms, new service infrastructures and architectures, as well as, flexible composition mechanisms.

Choreographies are an emergent Service Engineering approach to compose together and coordinate distributed services. Choreographies describe the interactions among the participant services from a global perspective. Choreographies model peer-to-peer communication by defining a multiparty protocol that, when put in place by the cooperating participant services, allows reaching the overall choreography goal in a fully distributed way. In this sense, service choreographies differ significantly from service orchestrations, in which one stakeholder (i.e., the orchestrator) centrally determines how to reach the goal through cooperation with other services. Future software systems will not be realized by orchestration only; they will also require choreographies. Indeed, services will be increasingly active entities that, communicating peer-to-peer, proactively make decisions and autonomously perform tasks according to their own imminent needs and the emergent global collaboration. Choreography does not rely on a central coordinator since each involved service knows exactly when

* Corresponding author.
E-mail addresses: marco.autili@univaq.it (M. Autili), paola.inverardi@univaq.it (P. Inverardi), massimo.tivoli@univaq.it (M. Tivoli).

https://doi.org/10.1016/j.scico.2017.10.010
0167-6423/

to execute its operations and with whom to interact. Thus, a choreography is a collaborative effort focusing on the exchange of messages among several participants to reach a common global goal.

Although conceptually different, at implementation level the two approaches may overlap to some extent. That is, a choreography might be realized as a set of distributed software entities implemented as BPEL[1] processes, which is the de-facto standard language for orchestrations. Thus, the technology used to realize them does not really matter. Rather, the real difference resides on their nature, purpose and, hence, on the way they are specified. That is, orchestration-oriented approaches consider a specification that is centralized and represents a form of centralized coordination, whose goal is to (re-)expose the resulting composed system in a way that it is amenable of further hierarchical composition. Instead, choreography-oriented approaches account for a specification that represents a form of distributed coordination of different participants collaborating to achieve a global common goal. Thus, the purpose is to exploit this specification to enable the prescribed collaboration in a fully-distributed way and let it emerges while the participants interact. The approach described in this paper provides an automated method to synthesize distributed software entities that, when needed, exchange coordination information to control the emerging interaction of third-party participants that, being reused, might not obey the specified choreography.

In the literature, many approaches have been proposed to deal with the foundational problems of checking choreography realizability, analyzing repairability of the choreography specification, verifying conformance, and enforcing realizability [2–12]. These approaches provide researchers with formal means to address fundamental aspects of choreographies. They are based on different interpretations of the choreography interaction semantics, in terms of both the subset of considered choreography constructs, and the used formal notations.

During the last decade, we have been working on choreographies within the context of the EU projects FP7 CHOReOS[2], and its follow-up H2020 CHOReVOLUTION[3]. Differently from most of the approaches mentioned above, our overall goal has always been to implement a complete choreography engineering process covering all the development activities, from specification to code synthesis, to automatic deployment and enactment on the Cloud. Furthermore, a peculiar characteristic is that the process has to support the effective reuse of existing third-party services. The outcome of our research and development activities is the CHOReVOLUTION Platform[4] together with an integrated development environment realized as a customization of the Eclipse platform called CHOReVOLUTION Studio[5].

With the aim to rigorously assess our approach to the synthesis of service choreographies [13–15], in this paper, we provide a complete formalization of the automated synthesis method for the distributed enforcement of choreography realizability implemented by the CHOReVOLUTION Studio. It takes as input a choreography specification in the form of a state machine and automatically generates a set of *Coordination Delegates* (CDs). CDs are additional software entities with respect to the choreography participants, and are synthesized in order to proxify and control the participant services interaction. When interposed among the services, the synthesized entities enforce the collaboration prescribed by the choreography specification. We prove that the synthesized CDs are correct by construction, meaning that the resulting choreographed system realizes the choreography specification, even when not realizable in principle. Our synthesis method deals with hybrid choreography participants that communicate synchronously and/or asynchronously. Leveraging the theoretical results on choreography realizability and its decidability in [4,11], CDs are able to handle asynchrony through 1-bounded FIFO queues.

The state machine that we use in this paper to rigorously specify choreographies is indeed an intermediate representation that the CHOReVOLUTION Studio internally uses to give a formal semantics to the set of BMPN2 Choreography Diagrams[6] constructs that our approach is able to deal with.

Summing up, the main contribution of this paper, with respect to our previous work within the CHOReOS and CHOReVOLUTION projects [13–15], concerns the rigorous formal assessment of the synthesis method, its underlying principles, and the proof of its correctness.

The paper is organized as follows. Section 2 overviews our approach providing the reader with its basic ingredients. Section 3 formalizes the approach and illustrates it at work on a simple, yet significant, explanatory example that stresses interesting distributed coordination problems. The example considers those difficult situations that make a choreography unrealizable. Furthermore, we state correctness of the synthesis method. Section 4 briefly discusses some practical tricky problems related to the actual implementation code of the synthesized CDs and their run-time. A discussion of related work is given in Section 5, and conclusions and future directions are presented in Section 6.

2. Approach overview

Fig. 1 shows an overview of our approach to the automatic synthesis of CDs that, when interposed among the participant services, control those interactions that need coordination in order to enforce the realizability of the specified choreography.

[1] http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.
[2] www.choreos.eu.
[3] www.chorevolution.eu.
[4] www.chorevolution.eu/bin/view/Documentation/.
[5] www.chorevolution.eu/bin/view/Documentation/Studio.
[6] http://www.omg.org/spec/BPMN/2.0/.

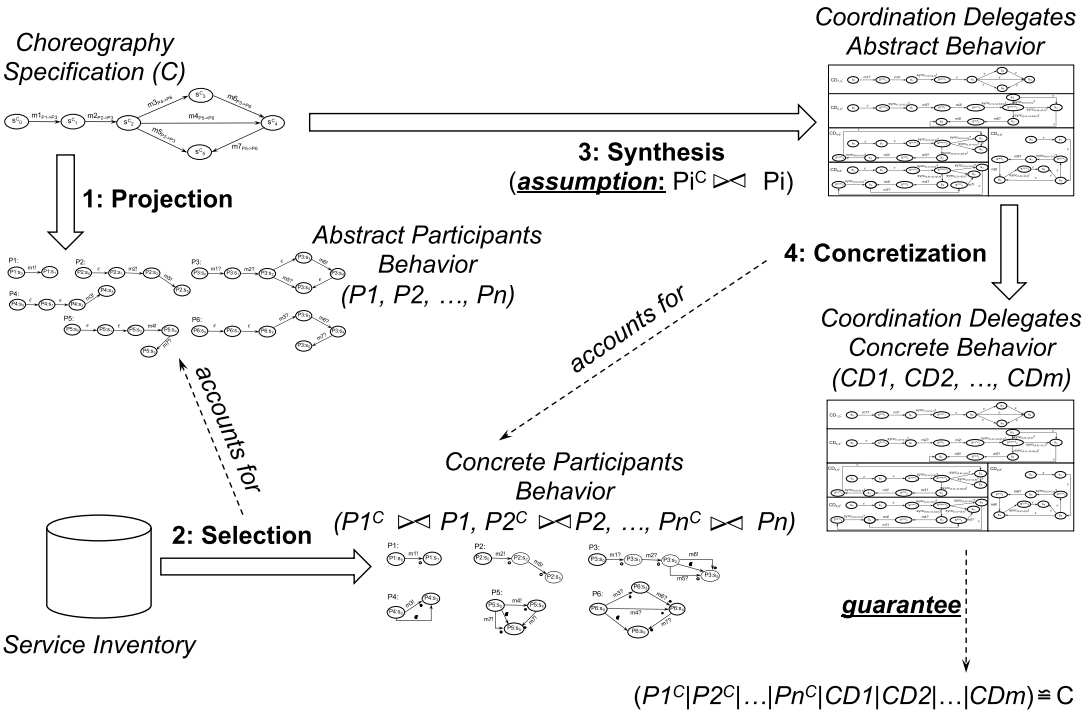

Fig. 1. Approach overview.

Our approach is organized into four steps that are performed in the following order: Projection, Selection, Synthesis, and Concretization.

1: Projection – It takes as input a choreography specification given in terms of a state machine where transitions model possibly simultaneous message exchanges among participants. As such, the choreography specification describes the way participants perform their interactions from a global perspective defining the (partial) order in which the message exchanges occur. Each single message exchange involves two participants: the sender and the receiver of the message. The specification abstracts from the way participants communicate to exchange messages, e.g., synchronous communication versus asynchronous one.

Out of the choreography, Projection generates a behavioral model for each participant. This model is a state machine where transitions model (sets of possibly simultaneous) actions sending or receiving message, or actions internal to the participant that are not observable from outside. Message send and receive are, instead, observable actions. Simultaneous actions serve to deal with parallel flows specified in the choreography and, hence, simultaneous executions. A projection represents the participant expected behavior according to the flows of message exchanges specified by the choreography. Being derived from the choreography specification, also this model abstracts from the type of the send and receive actions (synchronous or asynchronous). We call this model Abstract Participant Behavior.

2: Selection – We recall that our approach is reuse-oriented, meaning that it allows to enforce choreography realizability in contexts in which the choreography is not implemented from scratch but it is realized by reusing, as much as possible, third-party services published in a Service Inventory. Services are selected from the inventory to play the roles of the abstract participants in the choreography specification. As already said in Section 1, this calls for exogenous coordination of the selected concrete participants since, in general, we cannot access the participant code or change it.

A concrete service in the inventory comes with a behavioral specification of its interaction protocol. We call this model Concrete Participant Behavior. It is a state machine where transitions model (sets of possibly simultaneous) actions sending or receiving message, or internal actions. Similarly to the choreography specification, it can also specifies parallel flows that are joined afterwards. Differently from the Abstract Participant Behavior, for each transition, its type is specified: synchronous, asynchronous, or internal. That is, our approach does not impose constraints on the way concrete participants communicate, hence dealing with hybrid participants that can support both synchrony and asynchrony. For instance, a concrete participant could be a SOAP Web Service whose WSDL[7] interface defines both Request/Response (synchronous interaction) and One-way operations (asynchronous interaction). As it will be clearer in Section 3, to exchange messages asynchronously, concrete participants make use of bounded message queues. Our approach does not impose constraints on the size of the participants queues.

[7] www.w3.org/TR/wsdl.

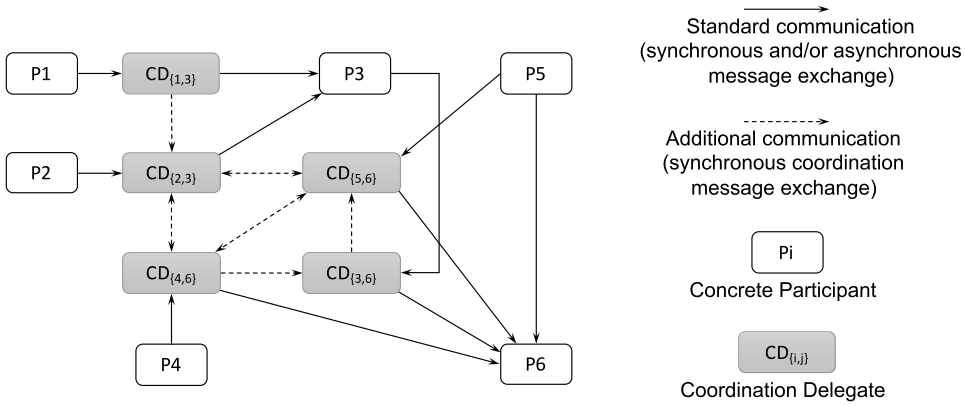

Fig. 2. Choreography-based architectural style (a sample instance of).

In order to select concrete participants that can suitably play the roles of abstract participants, our approach exploits a notion of behavioral refinement in order to automatically check whether the behavior of a concrete participant Pi^C is a refinement ($\bowtie$ in figure) of the behavior of an abstract participant Pi. In the best case, for each abstract participant, a suitable concrete participant is found in the inventory. Otherwise, it might be the case that the set of selected participants covers a subset of the choreography participants. In this case, the abstract behavior of the remaining choreography participants can support code generation activities to implement the missing concrete participants from scratch. Furthermore, the newly implemented concrete participant can be published in the inventory for possible future reuse.

An important consideration here is that, even in the case of limited reuse of third-party participants, our approach realizes separation of concern between the pure business logic implemented locally to each participant and the coordination logic needed for the realization of the choreography specification. This logic is automatically generated as a set of CDs (Synthesis step). Keeping the needed coordination logic separated from the business one saves developers from writing code that goes beyond the development of the pure business logic internal to single participants. This allows developers to realize the specified choreography, without requiring any specific attention to what concerns coordination aspects. This aspect has been appreciated by the industrial partners within the CHOReVOLUTION project since the approach permits to develop choreographies according to their daily development practices.

3 and 4: Synthesis and Concretization – The Synthesis step takes as input the choreography specification and automatically generates a set of CD Abstract Behavior models. Similarly to the Abstract Participant Behavior, each of them is a state machine where transitions model (sets of possibly simultaneous) actions sending or receiving message, or internal actions. These actions are related to the *standard* communication performed to achieve the choreography business logic. Differently from the Abstract Participant Behavior, there are also transitions modeling the synchronous exchange of coordination/synchronization messages. These actions model additional communication required to realize the coordination logic that is needed to enforce the realizability of the specified choreography. Standard communication takes place between a CD and the participant it controls and supervises, or directly among participants in case coordination is not required. When needed, additional communication messages are exchanged among the involved CD.

The Synthesis step is performed after a set of suitable concrete participants is obtained. Since the CD Abstract Behavior is generated out of the choreography specification, it abstracts from the way the supervised participants communicate (synchronously or asynchronously). This information will be added by the Concretization step that enriches the CD Abstract Behavior to achieve the so called CD Concrete Behavior.

For the set of synthesized CDs, correctness by construction means that when they are composed with the selected participants, the behavior of the resulting choreographed system realizes the specified choreography. That is, the generated CDs enforce by construction the realizability of the specified choreography. Leveraging results on choreography realizability and its decidability from the work in [4,11], to correctly deal with asynchrony, the concrete CDs (Concretization step) in the choreographed system make use of 1-bounded message queues.

According to a predefined architectural style, CDs are interposed only among the participants needing coordination. Fig. 2 shows an instance of the architectural style underlying our synthesis method.

CDs perform coordination (i.e., additional communication in the figure) of the participants interaction (i.e., standard communication in the figure) in a way that the resulting collaboration realizes the specified choreography. According to the type of actions performed by the concrete participants, standard communication can be synchronous or asynchronous. Additional communication is always synchronous. It is worth to note that CDs coordinate the interaction among the participants only when it is strictly needed for realizability enforcement purposes. That is, some participants are left free to communicate directly on those interactions that do not prevent the choreography realizability. Furthermore, depending on the specified choreography, CDs do not necessarily require to be connected one to each other.

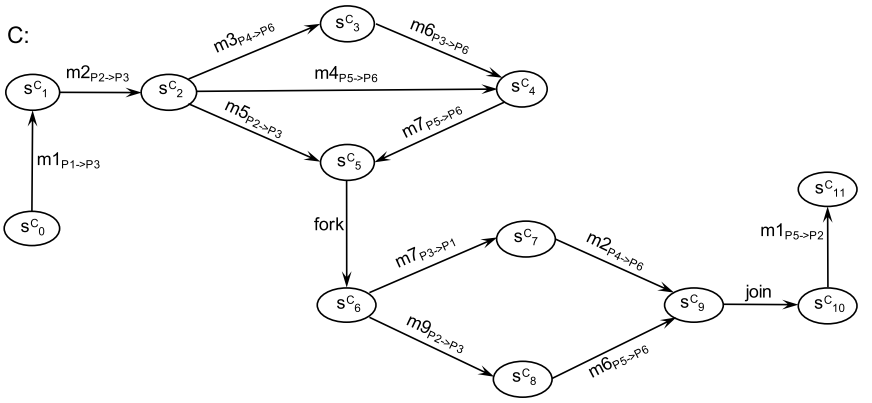

Fig. 3. Sample choreography specification.

3. Choreographies & synthesis

In this section, we give the definition of choreography specification (Section 3.1); abstract participant behavior and choreography projection (Section 3.2); concrete participant behavior and distributed system (Section 3.3) with synchronous and/or asynchronous communication. We also formalize our method for the automatic synthesis of CDs able to enforce choreography realizability by dealing with both synchrony and asynchrony, and with third-party (black-box) participants selected from the inventory (Section 3.4). Furthermore, we formalize the notion of choreographed system composing the selected participants with the synthesized CDs (Section 3.5). Finally, we state correctness of our CDs synthesis method (Section 3.6).

3.1. Choreography specification

A choreography is specified as a state machine modeling the interactions of a set of participants in terms of allowed message flows, i.e., possibly parallel sequences of observable message exchanges.

Definition 1 *(Choreography Specification).* A *Choreography Specification* $C = (\mathcal{P}, \mathcal{S}^C, s^C_0, \mathcal{L}, \mathcal{T}^C)$ is a state machine where $\mathcal{P}$ is a finite set of participants, $\mathcal{S}^C$ is a finite set of states with $s^C_0 \in \mathcal{S}^C$ being the initial state, and $\mathcal{L}$ is a finite set of message labels. $\mathcal{T}^C \subseteq \mathcal{S}^C \times ((\mathcal{P} \times \mathcal{L} \times \mathcal{P}) \cup \{(\bot, fork, \bot), (\bot, join, \bot)\}) \times \mathcal{S}^C$ is the transition relation, where: $(s^C_i, P, m, P', s^C_j) \in \mathcal{T}^C$ is an *exchange transition* modeling the sending of a message m from P to P'; $(s^C_i, \bot, fork, \bot, s^C_j) \in \mathcal{T}^C$ is a *fork transition* modeling the act of reaching the fork state s^C_j, and $(s^C_j, \bot, join, \bot, s^C_i) \in \mathcal{T}^C$ is a *join transition* modeling the act of leaving the join state s^C_j.

We write $s^C_i \xrightarrow{m_{P \to P'}} s^C_j$ to denote that $(s^C_i, P, m, P', s^C_j) \in \mathcal{T}^C$. Analogously, we write $s^C_i \xrightarrow{fork} s^C_j$ and $s^C_j \xrightarrow{join} s^C_i$ to denote that $(s^C_i, \bot, fork, \bot, s^C_j) \in \mathcal{T}^C$ and $(s^C_j, \bot, join, \bot, s^C_i) \in \mathcal{T}^C$, respectively.

For $s^C_i \in \mathcal{S}^C$, $|s^C_i|^{out}_{ex}$ indicates the number of exchange transitions outgoing from s^C_i and $|s^C_i|^{out}_{fork}$ indicates the number of outgoing fork transitions. Analogously, $|s^C_i|^{in}_{ex}$ and $|s^C_i|^{in}_{fork}$ indicate the number of incoming exchange transitions and incoming fork transitions, respectively. $|s^C_i|^{in}_{join}$ and $|s^C_i|^{out}_{join}$ are defined analogously.
States can be of the following types:

- s^C_i is a **plain state** if $|s^C_i|^{out}_{ex} \leq 1$, $|s^C_i|^{in}_{fork} = |s^C_i|^{out}_{join} = 0$, $|s^C_i|^{out}_{fork} \leq 1$ and $|s^C_i|^{in}_{join} \leq 1$
- s^C_i is a **branch state** if $|s^C_i|^{out}_{ex} \geq 2$, $|s^C_i|^{in}_{fork} = |s^C_i|^{out}_{join} = 0$, $|s^C_i|^{out}_{fork} \leq 1$ and $|s^C_i|^{in}_{join} \leq 1$
- s^C_i is a **fork state** if $|s^C_i|^{in}_{ex} = |s^C_i|^{in}_{join} = 0$, $|s^C_i|^{in}_{fork} = 1$, $|s^C_i|^{out}_{ex} + |s^C_i|^{out}_{fork} \geq 2$ and $|s^C_i|^{out}_{join} = 0$
- s^C_i is a **join state** if $|s^C_i|^{out}_{ex} = |s^C_i|^{out}_{fork} = 0$, $|s^C_i|^{out}_{join} = 1$, $|s^C_i|^{in}_{ex} + |s^C_i|^{in}_{join} \geq 2$ and $|s^C_i|^{in}_{fork} = 0$

The following properties hold:

- fork and join states are paired according to the usual pattern of balanced parenthesis in a math expression. That is, in our case, $B ::= fork\ B\ join\ B \mid fork\ join$ where B is a placeholder that can be recursively rewritten by $fork\ B\ join\ B$.
- Each sequence of transitions originating from a fork state leads to the corresponding paired join state.

Paired *fork/join states* together with *fork/join transitions* serve to deal with parallel sequences of message exchanges to be joined afterwards. Fork states model execution points where an arbitrary number of parallel flows originate and simultane-

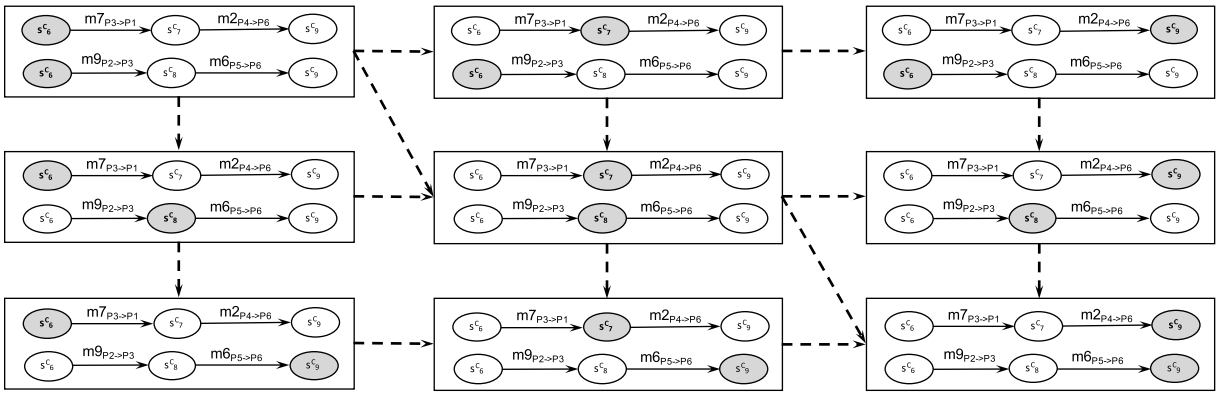

Fig. 4. All possible choreography frontiers between the paired fork/join states s_6^C and s_9^C.

ous message exchanges can be performed. Join states model synchronization points where an arbitrary number of parallel flows have to be joined. Thus, a choreography specification is essentially a labeled transition system suitably extended with fork/join transitions; alternative branches and sequences are naturally modeled through branch and plain states.

Fig. 3 shows a sample choreography. In this very simple choreography we have "artificially" included a set of interactions that will permit us to stress our approach at work on interesting distributed coordination problems. It contemplates those difficult situations that make the choreography unrealizable. We will see how our approach can deal with these situations in order to make the choreography realizable.

In Fig. 3, s_0^C, s_1^C, s_3^C, s_4^C, and s_5^C are plain states; s_2^C is a branch state; s_6^C and s_9^C are paired fork and join states, respectively; s_7^C, s_8^C, s_{10}^C and s_{11}^C are plain states. The possible interactions prescribed by the choreography in Fig. 3 are such that $P3$ receives the message $m1$ from $P1$, followed by $P3$ receiving the message $m2$ from $P2$. Then, when in the branch state s_2^C, three exclusive alternatives can be undertaken. One alternative accounts for $P6$ receiving $m3$ from $P4$, followed by $P6$ receiving $m6$ from $P3$, in turn followed by $P6$ receiving $m7$ from $P5$. The other two alternatives account for either $P6$ receiving $m4$ from $P5$, or $P3$ receiving $m5$ from $P2$. Each possible alternative path leads to the state s_5^C, from where the fork state s_6^C is reachable through a fork transition. Two parallel paths originate from s_6^C: (i) $P1$ receives the message $m7$ from $P3$, followed by $P6$ receiving $m2$ from $P4$; and (ii) $P3$ receives the message $m9$ from $P2$, followed by $P6$ receiving $m6$ from $P5$. All these paths join at the state s_9^C and then the choreography proceeds from s_{10}^C, hence finally terminating in s_{11}^C. Before terminating, $P2$ receives $m1$ from $P5$.

The *alphabet* Σ_C of a choreography $C = (\mathcal{P}, \mathcal{S}^C, s_0^C, \mathcal{L}, \mathcal{T}^C)$ is $\Sigma_C = \{m_{P\to P'} | P, P' \in \mathcal{P} \wedge m \in \mathcal{L}\} \cup \{fork, join\}$.

A *path* in a choreography C is a sequence of transitions of the form $s_i^C \xrightarrow{m_{P\to P'}} s_j^C$, $s_i^C \xrightarrow{fork} s_j^C$ or $s_i^C \xrightarrow{join} s_j^C$. The set of all paths in C is denoted by $Path(C)$.

A *sequence of interactions* in a choreography C is a sequence of actions over Σ_C.

A *trace* of a choreography C is a sequence of interactions from the start state. The set of all traces is denoted by $L(C)$ and it represents the *language* of C.

In Fig. 3, $s_1^C \xrightarrow{m2_{P2\to P3}} s_2^C \xrightarrow{m3_{P4\to P6}} s_3^C$ is a path from the state s_1^C to the state s_3^C, and $m2_{P2\to P3}\, m3_{P4\to P6}$ is the respective sequence of interactions. The sequence $m1_{P1\to P3}\, m2_{P2\to P3}\, m3_{P4\to P6}\, m6_{P3\to P6}\, m7_{P5\to P6}$ is a trace, as well as the trace $m1_{P1\to P3}\, m2_{P2\to P3}\, m3_{P4\to P6}\, m6_{P3\to P6}\, m7_{P5\to P6}\, fork\, m7_{P3\to P1}\, m2_{P4\to P6}\, join\, m1_{P5\to P2}$.

Concerning fork and join transitions, an important aspect of our approach is that we make use of the notion of *frontier* in order to identify the set of states, i.e., the frontier, in which the choreography can simultaneously be while parallel interactions are being performed. In Fig. 4, by distinguishing the two paths $s_6^C \xrightarrow{m7_{P3\to P1}} s_7^C \xrightarrow{m2_{P4\to P6}} s_9^C$ and $s_6^C \xrightarrow{m9_{P2\to P3}} s_8^C \xrightarrow{m6_{P5\to P6}} s_9^C$ between the paired fork/join states s_6^C and s_9^C, we highlight in grey all the possible choreography frontiers. In the figure, we also show all the possible frontier evolutions (see the dashed arrows), which can be easily derived by mapping the states constituting the frontiers to the corresponding states in the choreography.

Importantly, the notion of frontier allows our approach to deal with *real parallelism* as opposed to *abstract parallelism* only. That is, abstract parallelism is usually modeled though interleaving[8] of traces and permits to exchange a message at a time by following all the possible trace linearizations. Thus, abstract parallelism does not account for simultaneous message exchanges. Real parallelism permits the highest degree of parallelism since, in addition to the linearized interactions, also permits all the possible simultaneous exchanges, without imposing any order.

[8] Interleaving means merging two or more traces such that message exchanges from different traces may occur in any order in the resulting trace, while the message exchanges within the same traces retain their order. A trace resulting from this merge is usually called a linearization.

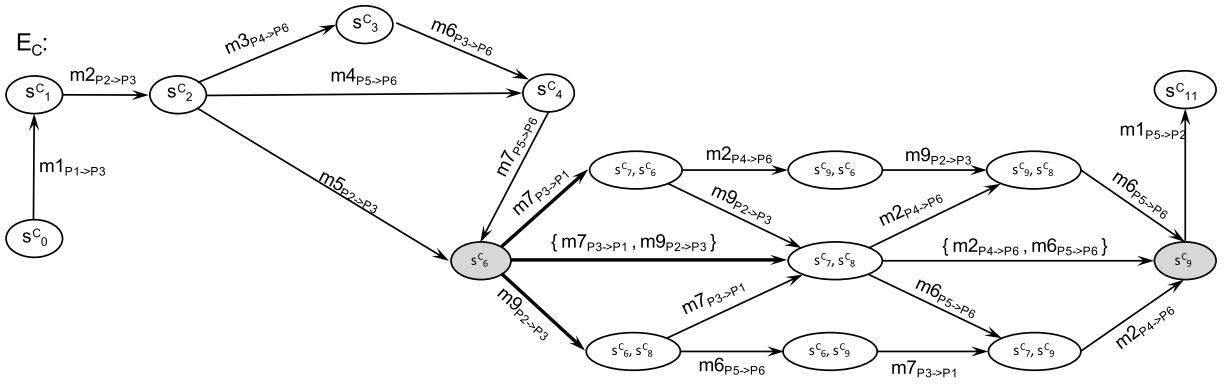

Fig. 5. Explicit-flow model of the sample choreography specification in Fig. 3.

Leveraging the notion of frontier, our method is able to automatically produce and intermediate representation of the choreography that, by transforming all the paired fork and join states, makes explicit all the possible interactions that can be performed through real parallelism. This representation is called choreography Explicit-flow Model (EfM).

Hereon, given a set Q, with 2^Q we denoted its power set. Informally, let $C = (\mathcal{P}, \mathcal{S}^C, s_0^C, \mathcal{L}, \mathcal{T}^C)$ be a choreography specification, its EfM is a state machine having the following structure: $E_C = (\mathcal{P}, \mathcal{S}', s_0^C, \mathcal{L}, \mathcal{T}', \mathcal{S}_J^F)$ where $\mathcal{S}'$ is contained in the union set of $\mathcal{S}^C$ and the set of states built by considering all the frontiers between the paired fork/join states in the choreography; $\mathcal{T}' \subseteq \mathcal{S}' \times 2^{(\mathcal{P}\times\mathcal{L}\times\mathcal{P}\cup\{\epsilon\})} \setminus \varnothing \times \mathcal{S}'$ is contained in the union set of $\mathcal{T}^C$ and the set of transitions suitably connecting the states in $\mathcal{S}'$; and $\mathcal{S}_J^F \subseteq \mathcal{S}' \times \mathcal{S}'$ is the set of paired fork/join states.

We write $s_i^C \xrightarrow{m_{P\to P'}} s_j^C$ to denote that $(s_i^C, \{(P, m, P')\}, s_j^C) \in \mathcal{T}'$. Analogously, $s_i^C \xrightarrow{\left\{m^1_{P_1\to P'_1},\ldots,m^k_{P_k\to P'_k}\right\}} s_j^C$ to denote that $(s_i^C, \{(P_1, m^1, P'_1), \ldots, (P_k, m^k, P'_k)\}, s_j^C) \in \mathcal{T}'$ for some $k > 1$.

Fig. 5 shows the EfM corresponding to the choreography in Fig. 3. Note that, accounting for the notion of frontier, we labeled the states between the paired fork/join states s_6^C and s_9^C with sets of state labels, hence explicitly representing the frontier in which the choreography can be while performing parallel interactions. Similarly, some transitions are also labeled with sets of message exchanges, meaning that the messages in the sets can be exchanged simultaneously. The paired fork/join states are filled in grey.

3.2. Abstract participant behavior and choreography projection

The behavior of a participant is modeled by a state machine describing all the interactions supported by the participant in terms of sent and received messages, whose exchange is observable from outside. Actions that cannot be observed from outside are modeled as internal actions.

Definition 2 *(Abstract Participant Behavior).* An *Abstract Participant Behavior* $B_k^A = (\mathcal{V}, v_0, \mathcal{M}, \delta^A, \mathcal{V}_J^F)$ for a participant P_k is a state machine where $\mathcal{V}$ is a finite set of states, $v_0 \in \mathcal{V}$ is the initial state, $\mathcal{M} = \mathcal{M}^? \cup \mathcal{M}^!$, $\mathcal{M}^?$ is a finite set of input messages and $\mathcal{M}^!$ is a finite set of output messages. $\delta \subseteq \mathcal{V} \times (2^{\mathcal{M}\cup\{\epsilon\}} \setminus \varnothing) \times \mathcal{V}$ is the transition relation. $\mathcal{V}_J^F \subseteq \mathcal{V}$ is the set of paired fork/join states. The transitions can have the following forms:

- $(v_i, \{m_1!\}, v_j)$ models a send action, i.e., the sending of a message $m_1 \in \mathcal{M}^!$ by P_k;
- $(v_i, \{m_1?\}, v_j)$ models a receive action, i.e., either the consumption by P_k of a message $m_1 \in \mathcal{M}^?$ from its message queue or the receiving of $m1$;
- either $(v_i, \{\epsilon\}, v_j)$ or $(v_i, \{\epsilon_k\}, v_j)$ for some $k > 0$ model an internal action of P_k;
- $(v_i, \{\alpha_1, \ldots, \alpha_n\}, v_j)$ for some $n > 1$ models the simultaneous execution of actions $\alpha_1, \ldots, \alpha_n$ of P_k.

We write $v_i \xrightarrow{\alpha} v_j$ to denote that $(v_i, \{\alpha\}, v_j) \in \delta^A$, $v_i \xrightarrow{\epsilon} v_j$ or $v_i \xrightarrow{\epsilon_k} v_j$ to denote that $(v_i, \{\epsilon\}, v_j) \in \delta^A$ or $(v_i, \{\epsilon_k\}, v_j) \in \delta^A$, and $v_i \xrightarrow{\{\alpha_1,\ldots,\alpha_n\}} v_j$ to denote that $(v_i, \{\alpha_1, \ldots, \alpha_n\}, v_j) \in \delta^A$.

The extension to an abstract participant behavior of the notions of path, alphabet, interactions sequence, trace, and language is straightforward.

The notion of participant behavior given by Definition 2 is abstract in the sense it does not account for whether the communication style adopted by the participant is synchronous or asynchronous. As it will be clear later, when the notions of concrete participant behavior and distributed system will be given in Section 3.3, a concrete (hybrid) participant can be

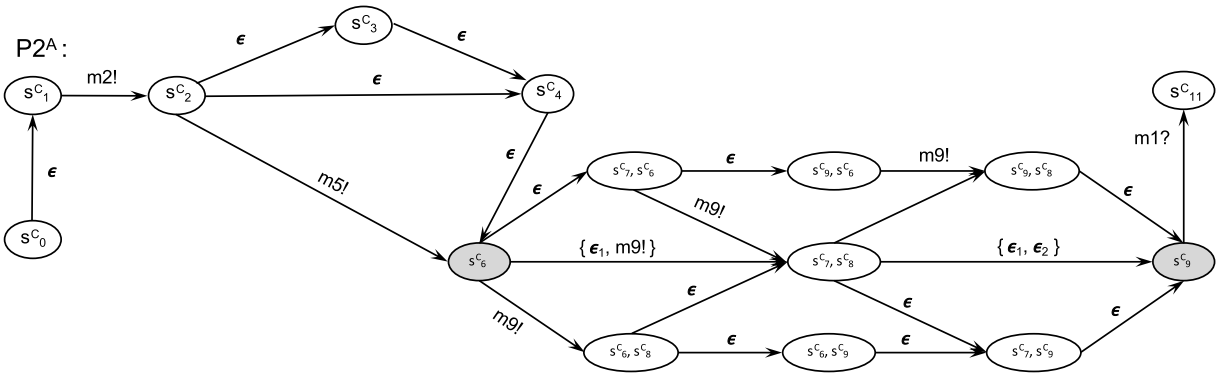

Fig. 6. Abstract behavior specification for the participant $P2$ achieved through projection.

composed with other participants and can interact both synchronously and asynchronously: synchronous message exchanges are handled through message queue of size zero; asynchronous message exchanges are handled through message queue of size greater than zero.

The abstract participant behavior is automatically produced out of the EfM of the Choreography Specification via projection as explained below. Thus, as such, it directly considers the notion of simultaneous actions (modeled as set of actions), hence dealing with parallel flows in an explicit way. Furthermore, it also considers paired fork/join states. Essentially, it is the EfM of the participant behavior.

Figs. 6 shows the abstract behavior specification $P2^A$ for the participant $P2$ in the choreography shown in Fig. 3. This specifications represent the *expected behavior* of a concrete participant that in the choreography shown in Fig. 3 can play the role of the abstract participant P_2. The expected behavior of a participant in a choreography can be generated through projection from the choreography specification as per the following definition.

Definition 3 *(Choreography projection).* Let $C = (\mathcal{P}, \mathcal{S}^C, s_0^C, \mathcal{L}, \mathcal{T}^C)$ be a choreography specification where $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ is the set of participants. Let $E_C = (\mathcal{P}, \mathcal{S}', s_0^C, \mathcal{L}, \mathcal{T}', \mathcal{S}_J^F)$ be the EfM of C, the projection of E_C on a participant $P_i \in \mathcal{P}$ is the abstract participant behavior $\Pi^C(P_i) = (\mathcal{S}', s_0^C, \mathcal{M}, \delta^A, \mathcal{S}_J^F)$ where $\mathcal{M} = \mathcal{M}^? \cup \mathcal{M}^!$ and the transition relation is such that:

- $\forall j \in [1..n]$ such that $j \neq i$: $(s_h, \{m!\}, s_k) \in \delta^A,\ m \in \mathcal{M}^!$ **if** $(s_h^C, P_i, m, P_j, s_k^C) \in \mathcal{T}'$,
  $(s_h, \{m?\}, s_k) \in \delta^A,\ m \in \mathcal{M}^?$ **if** $(s_h^C, P_j, m, P_i, s_k^C) \in \mathcal{T}'$
- $\forall x, y \in [1..n]$ such that $x, y \neq i$: $(s_h, \epsilon, s_k) \in \delta^A$ **if** $(s_h^C, P_x, m, P_y, s_k^C) \in \mathcal{T}'$
- $(s_h, \Omega, s_k) \in \delta^A$ **if** $(s_h^C, \{(P_{x_1}, m_1, P_{y_1}), \ldots, (P_{x_l}, m_l, P_{y_l})\}, s_k^C) \in \mathcal{T}'$ for some $l > 1$, **and**
  Ω is such that:
  $\forall q \in [1..l]$ such that $x_q, y_q \neq i$ then $\epsilon_q \in \Omega$,
  $\forall q \in [1..l]$ such that $y_q \neq i$ and $x_q = i$ then $m_q! \in \Omega$, **and**
  $\forall q \in [1..l]$ such that $x_q \neq i$ and $y_q = i$ then $m_q? \in \Omega$.

A choreography, whose specification $C = (\mathcal{P}, \mathcal{S}^C, s_0^C, \mathcal{L}, \mathcal{T}^C)$ where $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$, is *well-specified* if and only if for all $i \in [1..n]$ such that $\Pi^C(P_i) = (\mathcal{V}, v_0, \mathcal{M}^? \cup \mathcal{M}^!, \delta^A, \mathcal{V}_J^F)$, then $\mathcal{M}^? \cap \mathcal{M}^! = \varnothing$. Hereon, we assume to always deal with well-specified choreographies.

Beyond supporting the well-specified choreography check, the notion of choreography projection is used to suitably *select* from the inventory (see Section 2) concrete services that can play the role of an abstract participant in the choreography. As described below, the selection process exploit a notion of language refinement in order to check whether a concrete service "supports" the (expected) behavior of a given abstract participant.

In order to make the selection process more efficient, before performing the language refinement check, we perform a *minimization step* that, based on the following notion of language equivalence, removes superfluous ϵ transitions from the abstract participant behavior obtained through projection.

Language equivalence and minimization – The notion of language equivalence that we exploit to perform the minimization step is the weak trace equivalence [16] slightly adapted to our setting. That is, modulo internal actions (i.e., ϵ), the language of the concrete behavior must be the same as the language of the determinized abstract behavior. Concerning the minimization of simultaneous ϵ actions, they are treated as one single ϵ, and hence stepped over when possible. Furthermore, a set of actions that contains ϵ actions is considered as the same set of actions without the ϵ ones.

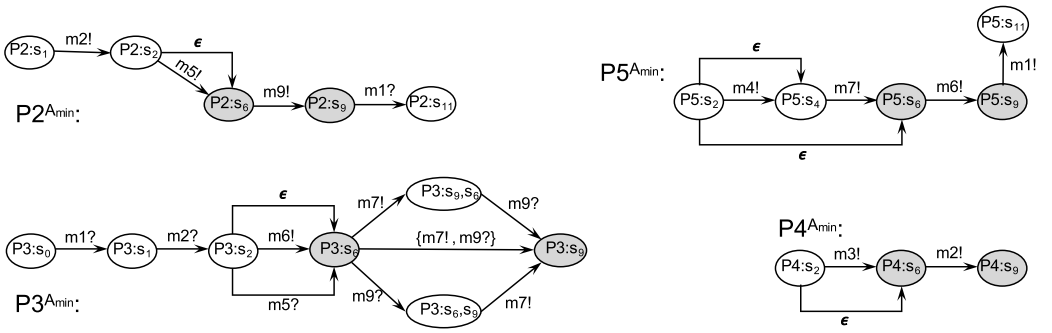

Fig. 7. Minimized abstract behavior specifications for the participants $P2$ to $P5$.

By applying the minimization step to the abstract behavior of $P2$ in Fig. 6, we achieve the corresponding minimized versions reported in Fig. 7, together with the ones for $P3$, $P4$ and $P5$.

When published in the inventory, a service comes with a description of its behavioral interface given in terms of a concrete participant behavior (see Definition 4 below). Based on the following notion of language refinement, if a service concrete behavior is a refinement of a participant abstract behavior as obtained via (minimized) projection, the service is selected to play the role of that participant in the choreography to be enforced.

Language refinement and selection – The notion of language refinement that we exploit to perform the selection step is based on the notion of automata-based language containment [17] (and reference therein), slightly adapted to our setting. Modulo internal actions (i.e., ϵ) and actions type (i.e., either synchronous or asynchronous), language containment checks whether the language of the EfM of a concrete participant behavior is the same as the language of an abstract participant behavior, except for the traces originating from paths between paired fork/join states where our notion of refinement only requires that, for each pair of fork/join states, the EfM of the concrete participant has at least one of these traces. The rationale behind is that parallel flows within paired fork/join states can be performed in different ways according to their EfM, and only one is enough to correctly realize the choreography. For this reason, we call it *alternating refinement*. We shall write $B_i^{A_{min}} \bowtie B_i^C$ to denote that B_i^C is an alternating refinement of $B_i^{A_{min}}$.

As it will be clear later, the notion of alternating refinement confers to our approach high flexibility while selecting concrete participants to realize the choreography since they are not required to support all the possible ways of realizing parallel flows, rather only one for each fork/join pair is required.

3.3. Concrete participant behavior and distributed system

In this section we formalize the notion of concrete participant behavior, and how their EfMs can be composed in parallel to build a hybrid distributed system with synchronous and/or asynchronous communications, possibly simultaneous. Note that different participants in the system can use message queues of different size.

Definition 4 *(Concrete Participant Behavior).* A *Concrete Participant Behavior* $B_k^C = (\mathcal{V}, v_0, \mathcal{M}, \delta^C)$ for a participant P_k is a state machine whose transitions are marked so as to distinguish asynchronous versus synchronous send/receive actions. Thus, $\delta^C \subseteq \mathcal{V} \times (\mathcal{M} \cup \{\epsilon, fork, join\}) \times \mathcal{V} \times \{\circ, \bullet, \epsilon\}$ and the transitions can have the following five forms:

- $(v_i, m_1!, v_j, \circ)$ models an asynchronous send action, i.e., the asynchronous sending of a message $m_1 \in \mathcal{M}^!$ by P_k;
- $(v_i, m_1?, v_j, \circ)$ models an asynchronous receive action, i.e., the consumption by P of a message $m_1 \in \mathcal{M}^?$ from its message queue;
- $(v_i, m_2!, v_j, \bullet)$ models a synchronous send action, i.e., the synchronous sending of a message $m_2 \in \mathcal{M}^!$ by P_k;
- $(v_i, m_2?, v_j, \bullet)$ models a synchronous receive action, i.e., the synchronous receiving by P_k of a message $m_2 \in \mathcal{M}^?$;
- $(v_i, \epsilon, v_j, \epsilon)$ models a generic internal action of P_k;
- $(v_i, fork, v_j, \epsilon)$ models an internal action of P_k specific to the act of reaching the fork state v_j;
- $(v_j, join, v_i, \epsilon)$ models an internal action of P_k specific to the act of leaving the join state v_j.

We write $v_i \xrightarrow{\alpha}_{\circ} v_j$ to denote that $(v_i, \alpha, v_j, \circ) \in \delta^C$, $v_i \xrightarrow{\beta}_{\bullet} v_j$ to denote that $(v_i, \beta, v_j, \bullet) \in \delta^C$, and $v_i \xrightarrow{\epsilon} v_j$ to denote that $(v_i, \epsilon, v_j, \epsilon) \in \delta^C$. We also write $v_i \xrightarrow{fork} v_j$ and $v_i \xrightarrow{join} v_j$ to denote that $(v_i, fork, v_j, \epsilon) \in \delta^C$ and $(v_i, join, v_j, \epsilon) \in \delta^C$, respectively.

The extension to a concrete participant behavior of the notions of plain, branch, fork, and join state (and related constraints as per Definition 1), as well as of path, alphabet, interactions sequence, trace, language and frontier is straightforward. An additional (practical) constraint for a concrete participant behavior is that all the transitions within paired fork

and join states must be either synchronous or asynchronous. Otherwise, its EfM model would not be properly defined due to the presence of simultaneous actions mixing synchronous and asynchronous communication.

Before formalizing the notion of hybrid system, it is worth to note that mixing synchronous actions and asynchronous ones is useful in several contexts related to the practice of choreography development. For instance, a concrete participant behavior could be used to model the behavioral protocol of a web service defining, through its WSDL interface, both request/response operations (synchronous send and receive actions) and one-way operations (asynchronous send and receive actions).

Definition 5 *(Hybrid System Behavior).* Let $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ be a set of participants, let $q_1, \ldots, q_n$ be the maximum size values predefined for their message queues $Q_1, \ldots, Q_n$ respectively, and let $B_i^C = (\mathcal{V}_i, v_{0i}, \mathcal{M}_i, \delta_i^C, \mathcal{V}_{iJ}^F)$ be the EfM of the concrete behavior of P_i such that $\forall i, j : i \neq j \implies \mathcal{M}_i^? \cap \mathcal{M}_j^? = \mathcal{M}_i^! \cap \mathcal{M}_j^! = \varnothing$. The behavior of a hybrid distributed system composing the participants in $\mathcal{P}$ is modeled as a state machine $H = (\mathcal{P}, \mathcal{S}, s_0, \mathcal{M}, \Delta)$ where $\mathcal{S} \subseteq \mathcal{V}_1 \times \mathcal{Q}_1 \times \ldots \times \mathcal{V}_n \times \mathcal{Q}_n$ such that $\forall i \in [1 \ldots n] : \mathcal{Q}_i \subseteq (\mathcal{M}_i^?)^{q_i}$, $s_0 \in \mathcal{S}$ such that $s_0 = (v_{01}, \epsilon, \ldots, v_{0n}, \epsilon)$ with ϵ denoting the empty string, and $\mathcal{M} = \bigcup_{i=1}^{n} \mathcal{M}_i$.

For $s = (v_1, Q_1, \ldots, v_n, Q_n) \in \mathcal{S}$ and $s' = (v'_1, Q'_1, \ldots, v'_n, Q'_n) \in \mathcal{S}$, the transition relation $\Delta \subseteq \mathcal{S} \times 2^{(\mathcal{P} \times \mathcal{M} \times \mathcal{P} \cup \{\epsilon\})} \setminus \varnothing \times \mathcal{S} \times \{\bullet, \circ, \epsilon\}$ is such that:

Rule 1 – Asynchronous send-receive actions and message consumptions

$s \xrightarrow{\{x_1, \ldots, x_l\}}_{\circ} s' \in \Delta$ if
$\exists i_1, \ldots, i_l, j_1, \ldots, j_l \in [1..n] : \ \exists m^1 \in \mathcal{M}_{i_1}^! \cap \mathcal{M}_{j_1}^?, \ldots, m^l \in \mathcal{M}_{i_l}^! \cap \mathcal{M}_{j_l}^? :$
$v_{i_1} \xrightarrow{I_{i_1}^? \cup I_{i_1}^!}_{\circ} v'_{i_1} \in \delta_{i_1}^C, \ldots, v_{i_l} \xrightarrow{I_{i_l}^? \cup I_{i_l}^!}_{\circ} v'_{i_l} \in \delta_{i_l}^C,$
$v_{j_1} \xrightarrow{I_{j_1}^? \cup I_{j_1}^!}_{\circ} v'_{j_1} \in \delta_{j_1}^C, \ldots, v_{j_l} \xrightarrow{I_{j_l}^? \cup I_{j_l}^!}_{\circ} v'_{j_l} \in \delta_{j_l}^C,$
$I_{i_1}^!, \ldots, I_{i_l}^! \in 2^{\{m^1!, \ldots, m^l!\}}$, $I_{j_1}^?, \ldots, I_{j_l}^? \in 2^{\{m^1?, \ldots, m^l?\}}$, $I_{i_1}^! \cap I_{i_1}^? = \varnothing, \ldots, I_{j_l}^! \cap I_{j_l}^? = \varnothing,$
$\forall s, t \in [1..l] : m^s? \in \bigcup_{h=1}^{l} (I_{i_h}^? \cup I_{j_h}^?), m^t! \in \bigcup_{h=1}^{l} (I_{i_h}^! \cup I_{j_h}^!) \implies Q_{j_s}^{s-1} = m^s Q_{j_s}^s, x_s = \epsilon_s,$
$|Q_{j_t}^{t-1}| < q_{j_t}$, $Q_{j_t}^t = Q_{j_t}^{t-1} m^t$, $x_t = m^t_{P_{i_t} \to P_{j_t}}$, (where $|Q_r|$ is the length of Q_r, $Q_r^0 = Q_r$, and $Q_r^l = Q'_r$)
$\forall k \in [1..n] : \forall t \in [1..l] : k \neq i_t, j_t \implies Q'_k = Q_k$, $v'_k = v_k$, and
$\forall k \in [1..n] : \forall t \in [1..l] : k \neq j_t \implies Q'_k = Q_k$

Rule 2 – Synchronous send-receive actions and message consumptions

$s \xrightarrow{\{x_1, \ldots, x_l\}}_{\bullet} s' \in \Delta$ if
$\exists i_1, \ldots, i_l, j_1, \ldots, j_l \in [1..n] : \ \exists m^1 \in \mathcal{M}_{i_1}^! \cap \mathcal{M}_{j_1}^?, \ldots, m^l \in \mathcal{M}_{i_l}^! \cap \mathcal{M}_{j_l}^? :$
$v_{i_1} \xrightarrow{I_{i_1}^? \cup I_{i_1}^!}_{\bullet} v'_{i_1} \in \delta_{i_1}^C, \ldots, v_{i_l} \xrightarrow{I_{i_l}^? \cup I_{i_l}^!}_{\bullet} v'_{i_l} \in \delta_{i_l}^C,$
$v_{j_1} \xrightarrow{I_{j_1}^? \cup I_{j_1}^!}_{\bullet} v'_{j_1} \in \delta_{j_1}^C, \ldots, v_{j_l} \xrightarrow{I_{j_l}^? \cup I_{j_l}^!}_{\bullet} v'_{j_l} \in \delta_{j_l}^C,$
$I_{i_1}^!, \ldots, I_{i_l}^! \in 2^{\{m^1!, \ldots, m^l!\}}$, $I_{j_1}^?, \ldots, I_{j_l}^? \in 2^{\{m^1?, \ldots, m^l?\}}$, $I_{i_1}^! \cap I_{i_1}^? = \varnothing, \ldots, I_{j_l}^! \cap I_{j_l}^? = \varnothing,$
$\forall t \in [1..l] : m^t? \in \bigcup_{h=1}^{l} (I_{i_h}^? \cup I_{j_h}^?), m^t! \in \bigcup_{h=1}^{l} (I_{i_h}^! \cup I_{j_h}^!) \implies x_t = m^t_{P_{i_t} \to P_{j_t}},$
$\forall k \in [1..n] : Q'_k = Q_k$ and $\forall k \in [1..n] : \forall t \in [1..l] : k \neq i_t, j_t \implies v'_k = v_k$

Rule 3 – Internal actions (i.e., ϵ)

$s \xrightarrow{\epsilon} s' \in \Delta$ if $\exists i \in [1..n] : \ v_i \xrightarrow{\epsilon} v'_i \in \delta_i^C,$
$\forall k \in [1..n] : Q'_k = Q_k$ and $\forall k \in [1..n] : k \neq i \implies v'_k = v_k$

The extension to a hybrid system behavior of the notions of path, alphabet, interactions sequence, trace, and language is straightforward.

According to Definition 5, in a composed system, asynchronous send and receive actions together model an asynchronous interaction where the receiving participant uses a bounded FIFO queue. In particular, an asynchronous send models a participant sending a request message to a receiving participant. The message is appended to the message queue of the receiving participant, unless the queue is full. An asynchronous receive action models a participant consuming a message from the head of its queue, unless either the queue is empty or the head contains a message different from the one to be consumed. Thus, asynchronous send actions are blocking only if the queue of the receiving participant is full. Analogously, asynchronous receive actions are blocking only if either the participant queue is empty or a message different from the one to be consumed is on the head of the queue.

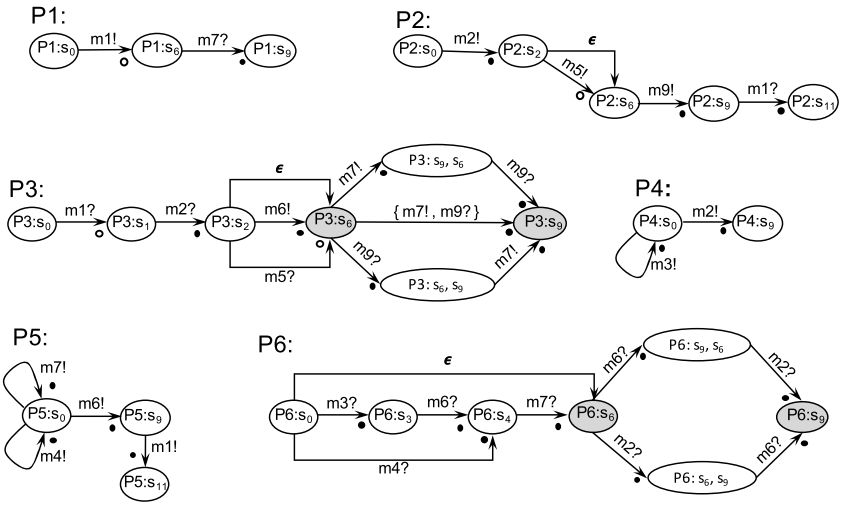

Fig. 8. Explicit-flow models of the concrete behavior specifications for the participants $P1$ to $P6$.

Synchronous send and receive actions together models a synchronous interaction where the sent message is immediately consumed by the receiving participant. Thus, no queue is involved while performing synchronous actions.

Multiple actions, i.e., the ones modeled by transitions labeled with set of actions, are executed simultaneously. Thus, a participant exchanging multiple messages with other participants leads the parallel composition to exhibit the corresponding multiple exchanges of messages, hence reaching a global state modeling the simultaneous progress of all the involved participants.

Finally, the participants can step over internal actions (i.e., ϵ) independently. Note that, being local to the single participants, asynchronous receive actions are not observable in the system and hence become ϵ actions.

In their simpler case, i.e., without considering simultaneous multiple actions, asynchronous send and asynchronous receive actions are treated according to the following basic rules:

Asynchronous send action

$$s \xrightarrow{m_{P_i \to P_j}}_{\circ} s' \in \Delta \text{ if } \exists i, j \in [1..n] : \; m \in \mathcal{M}_i^! \cap \mathcal{M}_j^?,\; v_i \xrightarrow{m!}_{\circ} v_i' \in \delta_i^C,$$
$$|Q_j| < q_j,\; Q_j' = Q_j m, \quad (|Q_j| \text{ is the length of } Q_j)$$
$$\forall k \in [1..n] : k \neq j \implies Q_k' = Q_k \text{ and } \forall k \in [1..n] : k \neq i \implies v_k' = v_k$$

Asynchronous receive action and message consumption

$$s \xrightarrow{\epsilon} s' \in \Delta \text{ if } \exists j \in [1..n] : \; m \in \mathcal{M}_j^?,\; v_j \xrightarrow{m?}_{\circ} v_j' \in \delta_j^C,$$
$$Q_j = m Q_j',$$
$$\forall k \in [1..n] : k \neq j \implies Q_k' = Q_k \text{ and } \forall k \in [1..n] : k \neq j \implies v_k' = v_k$$

Essentially, Rule 1 of Definition 5 combines the two basic rules above by generalizing them to the case of multiple participants involved simultaneously in different asynchronous message exchanges, hence accounting for several simultaneous asynchronous actions.

Analogously, in its simpler case, synchronous send and receive actions are accounted for by the following basic rule:

Synchronous send-receive actions and message consumptions

$$s \xrightarrow{m'_{P_j \to P_i}}_{\bullet} s' \in \Delta \text{ if } \exists i, j \in [1..n] : \; m' \in \mathcal{M}_j^! \cap \mathcal{M}_i^?,\; v_j \xrightarrow{m'!}_{\bullet} v_j' \in \delta_j^C,\; v_i \xrightarrow{m'?}_{\bullet} v_i' \in \delta_i^C,$$
$$\forall k \in [1..n] : Q_k' = Q_k \text{ and } \forall k \in [1..n] : k \neq i, j \implies v_k' = v_k$$

Rule 2 of Definition 5 generalizes the above rule to the case of several participants involved simultaneously in synchronous message exchanges.

Figs. 9 and 10 show two excerpts of the hybrid system obtained by composing the set of EfMs of concrete participants $P1$ to $P6$ shown in Fig. 8. These participants have been selected from the inventory so to play the respective roles of the choreography C in Fig. 3. In fact, according to the notion of language containment previously introduced, their concrete behavior is an alternating refinement of $\Pi^C(Pi)$, $i \in [1..6]$, respectively. In particular, the concrete participants $P1$, $P2$, and $P3$, whose explicit-flow models are shown in Fig. 8, have the same language as the one of the corresponding abstract ones shown in Fig. 7. Instead, the languages of abstract participants $P4$ and $P5$ are strictly included in the languages of

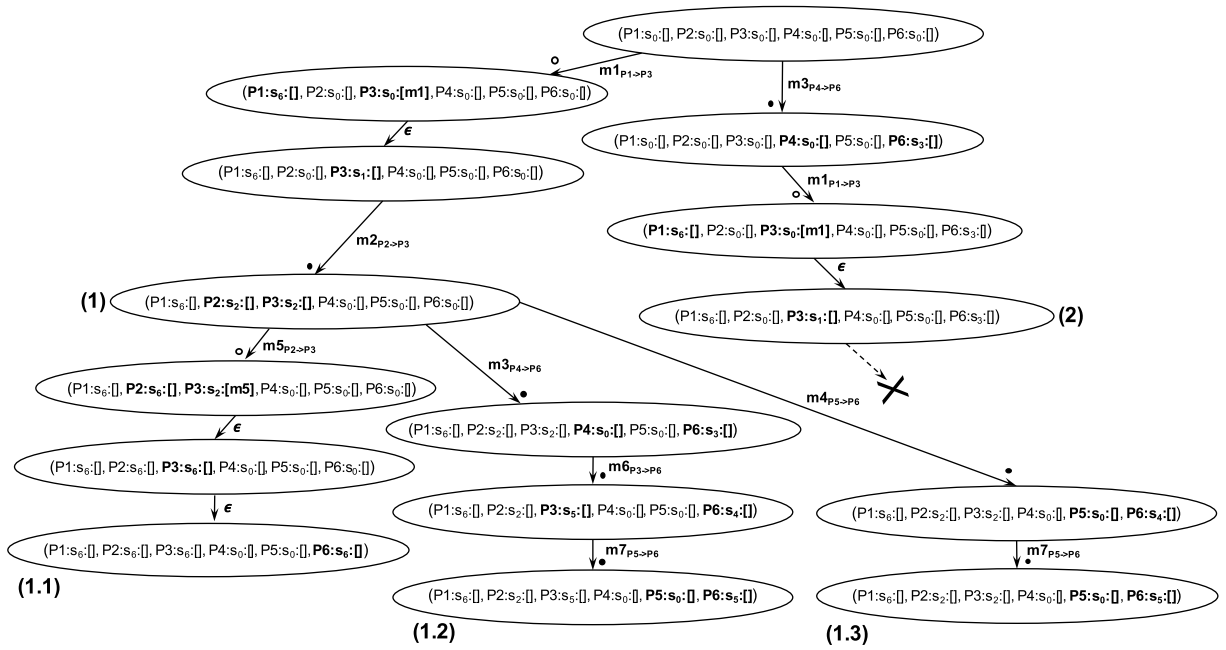

Fig. 9. An excerpt of the hybrid system composing the participants $P1$–$P6$ (1-bounded queues).

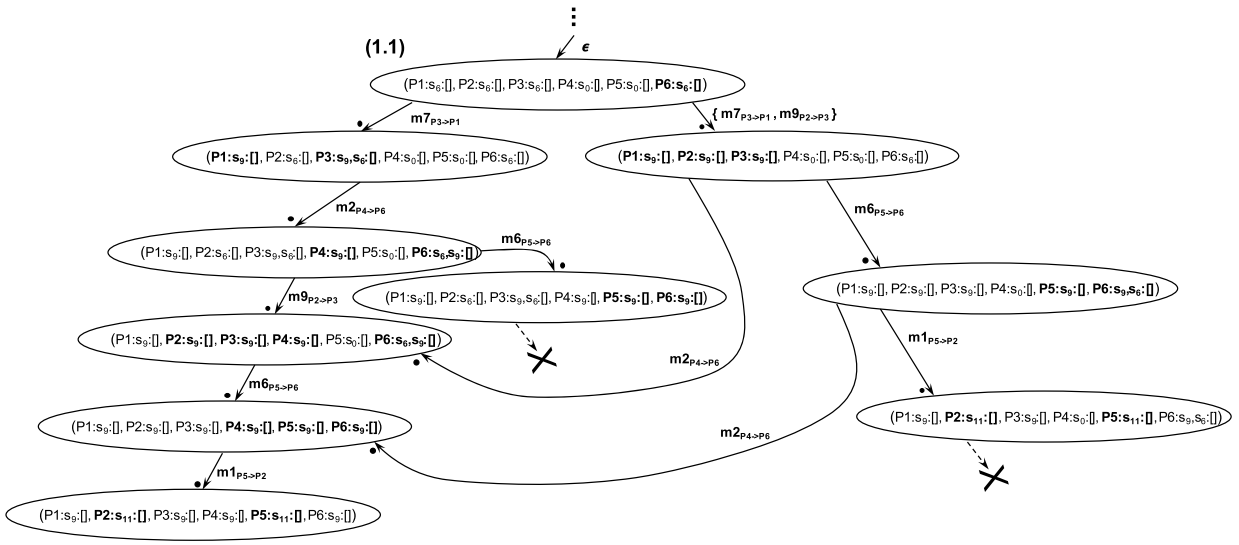

Fig. 10. An excerpt of the hybrid system starting from the state marked with **(1.1)** in Fig. 9.

corresponding concrete participants, meaning that the concrete participants support more interactions with respect to the ones required to realize the choreography. These interactions will be filtered and coordinated by the coordination delegates. Furthermore, the EfM of the concrete participant $P6$ is an alternating refinement of the one of the corresponding abstract participant. In fact, it has the same language of the one of the abstract participant behavior, except for the traces that ends with the simultaneous actions $\{m2?, m6?\}$, which are not present in the language of the concrete participant behavior. Note that, these simultaneous actions would be performed in the abstract participant behavior between the paired fork/join states $P6{:}s_6$ and $P6{:}s_9$ and, hence, this conforms to our notion of refinement.

In the graphical representation of Fig. 8, we have named the states so to facilitate the mapping of the choreography states in Fig. 3 to the states of the hybrid system behavior resulting from the composition in Figs. 9 and 10. For each participant Pi, its states are labeled as either *Pi:state-name* or, in the case of simultaneous actions, *Pi:state-name-1, ..., state-name-n*. In general, the hybrid system behavior resulting from the composition of a set of participants has a finite number of states as long as the concrete behaviors of all the participants are finite state and have bounded queues.

In the example, the messages $m2$, $m3$, $m4$, $m6$, $m7$, $m9$ are exchanged synchronously; depending on the participant sending it, $m1$ can be exchanged either synchronously or asynchronously; $m5$ is exchanged asynchronously; finally, due to the specified concurrent flows, $m7$ can be also exchanged simultaneously with $m9$ and $m6$ simultaneously with $m2$. Moreover, in the hybrid system in Figs. 9 and 10, asynchronous interactions are handled through messages queues all having maximum size equals to 1, i.e., the participants $P1$, $P2$ and $P3$ are all assigned 1-bounded queues; whereas, the participant $P4$, $P5$ and $P6$ are all assigned 0-bounded queues (they exchange only synchronous messages). The participant $P3$ is indeed a hybrid participant since it exchanges $m6$ synchronously, and $m1$ and $m5$ asynchronously, as well as participants $P1$ and $P2$. In the figure, message queues are denoted as [...]. Initially, all the queues are empty, hence []. State changes and queue updates are highlighted in bold step by step.

Following the branch on the left-hand side of Fig. 9, three different paths can be undertaken from the state marked with **(1)**. The corresponding traces are:

$m1_{P1\to P3} \in m2_{P2\to P3}\, m5_{P2\to P3} \in \in$

$m1_{P1\to P3} \in m2_{P2\to P3}\, m3_{P4\to P6}\, m6_{P3\to P6}\, m7_{P5\to P6}$

$m1_{P1\to P3} \in m2_{P2\to P3}\, m4_{P5\to P6}\, m7_{P5\to P6}$

It is easy to see that these three traces are all allowed by the choreography EfM in Fig. 5. Contrariwise, all the traces (not completely shown in Fig. 9) traversing the state marked with **(2)** are not allowed by the choreography. Furthermore, by continuing from the state marked with **(1.1)** in Fig. 10, two traces that are not allowed can be exhibited by the parallel composition:

$t1 = \ldots\, m7_{P3\to P1}\, m2_{P4\to P6}\, m6_{P5\to P6}\, \ldots$

$t2 = \ldots\, \{m7_{P3\to P1}, m9_{P2\to P3}\}\, m6_{P5\to P6}\, m1_{P5\to P2}\, \ldots$

Trace $t1$ is not allowed since, while executing the concurrent flows, the choreography EfM requires $m6$ to be exchanged only after $m9$ is exchanged. Trace $t2$ is not allowed since it corresponds to participant $P2$ exchanging $m1$ with $P5$ without waiting the concurrent execution of participant $P6$ reaching the specified join state. Thus, according to the notion of choreography realizability given in [4,11], the choreography in Fig. 3 is not realizable. Intuitively, this basically means that if we let the selected participants to interact in an uncontrolled way in the composed system, they can perform interactions that are not permitted by the choreography specification. The considerations below follow.

Choreography realizability enforcement – Given a choreography C, whose EfM is E_C, let H^C be the *corresponding hybrid system* composing a set of concretized[9] participant behaviors obtained through (minimized) projection from E_C. Leveraging results on choreography realizability from the work in [4,11], we have that $L(E_C) \neq L(H_1^C) \implies L(E_C) \subset L(H_1^C)$, with H_1^C being the system corresponding to H^C where each participant is assigned a 1-bounded queue. In other words, $L(H_1^C)$ contains at least a trace that is not contained in $L(E_C)$.

The implication $L(E_C) \neq L(H_1^C) \implies L(E_C) \subset L(H_1^C)$ comes from the formal definition of realizability in [4,11] according to which a choreography C *is realizable* if and only if $L(C) = L(H_1^C)$, i.e., $L(E_C) = L(H_1^C)$ in our setting. Informally, the implication basically means that if the choreography C is not realizable, then the corresponding system H_1^C and (hence) H^C are able to perform more interaction sequences than the ones specified by E_C.

In the following we will show that, when the behavior of the concrete participants is an alternating refinement of the abstract behavior of the participants as obtained via projection, it is possible to automatically synthesize a set of 1-bounded CDs that enforce the realization of the choreography. When interposed among the concrete participants in the system, CDs perform exogenous coordination by *restricting* the interaction sequences in the resulting choreographed system CS in such a way that $C \cong CS$, i.e., $L(E_C) = L(CS)$ modulo ϵ-actions and the traces in $L(E_C)$ originating from paths between paired fork/join states for which $L(CS)$ is only required to contain, for each pair of fork/join states, at least one of these traces.

3.4. Coordination delegates synthesis

CDs are realized as (additional) hybrid participants that, in the overall choreographed systems, handle *standard communication* through 1-bounded message queues, and perform *additional communication* to exchange coordination information messages synchronously.

Interposing a CD between two participants, when needed, means to decouple the interactions of the participants only for those messages that require coordination. Thus, later in the paper we will define the behavior of a choreographed system by requiring that the architectural configuration associated to it must be specified according to the architectural style in Fig. 2. In the formalization, this allows us to achieve decoupling without relabeling the transition labels for those messages that require the interposition of CDs. Coordination information is extracted by identifying independent sequences, independent branches, and parallel flows in the choreography specification:

- *independent sequences* of message exchanges can be identified when the EfM of the choreography specification C includes a path where two different (sender) participants send two messages and, in the corresponding hybrid system H^C, no dependency can be established between the sender of the second message and the sender of the first message.

[9] Whatever communication style is chosen, i.e., synchronous, asynchronous, or hybrid.

- *independent branches* can be identified when the EfM includes branch states from which different alternative transitions can be undertaken from at least two different (sender) participants. Again, in the corresponding hybrid system H^C, no dependency can be established between the sender in a branch and the sender(s) in the other branch(es).
- *parallel flows* can be identified through paired fork/join states of the EfM.

For instance, in the choreography EfM shown in Fig. 5, $s_0^C \xrightarrow{m1_{P1\to P3}} s_1^C \xrightarrow{m2_{P2\to P3}} s_2^C$ is a path from the state s_0^C to the state s_2^C that identifies the independent sequence of interaction $m1_{P1\to P3}\, m2_{P2\to P3}$. The branch state s_2^C identifies the independent branches $s_2^C \xrightarrow{m3_{P4\to P6}} s_3^C$, $s_2^C \xrightarrow{m4_{P5\to P6}} s_4^C$ and $s_2^C \xrightarrow{m5_{P2\to P3}} s_6^C$. The flows within the paired fork/join states s_6^C and s_9^C identify all the possible ways to realize the two parallel sequences of interaction $s_6^C \xrightarrow{m7_{P3\to P1}} s_7^C \xrightarrow{m2_{P4\to P6}} s_9^C$ and $s_6^C \xrightarrow{m9_{P2\to P3}} s_8^C \xrightarrow{m6_{P5\to P6}} s_9^C$ in the choreography. For instance, the sequence of simultaneous actions $s_6^C \xrightarrow{\{m7_{P3\to P1}, m9_{P2\to P3}\}} (s_7^C, s_8^C) \xrightarrow{\{m2_{P4\to P6}, m6_{P5\to P6}\}} s_9^C$, or the sequence $s_6^C \xrightarrow{m7_{P3\to P1}} (s_7^C, s_6^C) \xrightarrow{m9_{P2\to P3}} (s_7^C, s_8^C) \xrightarrow{m6_{P5\to P6}} (s_7^C, s_9^C) \xrightarrow{m2_{P4\to P6}} s_9^C$.

From the examples above, it is clear that both independent sequences and independent branches require coordination; whereas, no coordination for dependent sequences and dependent branches is needed since only one sender participant is involved and, the possible resulting interactions depend only on it. Concerning parallel flows, coordination is required to ensure that the participants interact by following at least one of the possible sequences made explicit within each fork/join pair in the EfM.

To ease the understanding of the CDs synthesis method and to better present it at work on our explanatory example, we first formalize how CDs are automatically synthesized in order to coordinate the interaction of the participants involved in independent sequences and independent branches (see Section 3.4.1). Then, we enhance the previous formalization in order to show how the method automatically synthesizes CDs that are able to also deal with the distributed coordination of the participants involved in the execution of parallel flows (see Section 3.4.2).

3.4.1. Dealing with independent sequences and independent branches

Definition 6 below brings together these basic considerations, and formalizes the synthesis of CDs by distinguishing between 1) *independent sequences across dependent branches* and 2) *independent sequences across independent branches*. Still referring to the choreography EfM in Fig. 5, $s_1^C \xrightarrow{m2_{P2\to P3}} s_2^C \xrightarrow{m3_{P4\to P6}} s_3^C$ identifies the independent sequence $m2_{P2\to P3}\, m3_{P4\to P6}$ across the independent branches identified by the branch state s_2^C.

For the sake of readability, in Definition 6, when synthesizing the behavior of a CD $CD_{\{x,y\}} = (\mathcal{V}_{\{x,y\}}, s_0, \mathcal{M}_{\{x,y\}}, \delta_{\{x,y\}})$, we shall write $s \xrightarrow{\alpha} s' \xrightarrow{\alpha'} s'' \ldots s^{k-1} \xrightarrow{\alpha^{k-1}} s^k \in Path(CD_{\{x,y\}})$ to also assert that $s, s', s'', \ldots, s^{k-1}, s^k \in \mathcal{V}_{\{x,y\}}$ and $\alpha, \alpha', \ldots, \alpha^{k-1} \in \mathcal{M}_{\{x,y\}}$. The two rules in Definition 6 have the following form:

[name of the rule]
(i) [assertion 1],
(ii) [assertion 2],
(...) [... ...]
if *[condition]*

Definition 6 *(CDs Abstract Behavior Synthesis).* Let $C = (\mathcal{P}, \mathcal{S}^C, s_0^C, \mathcal{L}, \mathcal{T}^C)$ be a choreography specification where $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ is the set of participants. Let $E_C = (\mathcal{P}, \mathcal{S}', s_0^C, \mathcal{L}, \mathcal{T}', \mathcal{S}_J^F)$ be the EfM of C. The set $\mathcal{CD}^C \subseteq \left\{CD_{\{i,j\}} \mid i, j \in [1..n]\right\}$ of the CD abstract behaviors that are needed to enforce the realization of C is such that for all $CD_{\{x,y\}} = (\mathcal{V}_{\{x,y\}}, s_0, \mathcal{M}_{\{x,y\}}, \delta_{\{x,y\}}) \in \mathcal{CD}^C$ we have that $\forall s_i^C \in \mathcal{S}'$: $s_i \in \mathcal{V}_{\{x,y\}}$(i.e., each CD inherits its states from E_C) and:

1. Coordination of independent sequences across dependent branches

(i) $CD_{\{x,y\}} \in \mathcal{CD}^C$, $\forall t \in [1..q]$: $CD_{\{z,wt\}} \in \mathcal{CD}^C$,

(ii) $s_i \xrightarrow{m?} s_i^{mid} \xrightarrow{m!} s_j \xrightarrow{\mathtt{sync}_{\{x,y\}\to\{z,w1\}\{z,w2\}\ldots\{z,wq\}}!} s_j^{sync} \in Path(CD_{\{x,y\}})$

(iii.a) $\forall t \in [1..q]$: $s_j^{sync} \xrightarrow{\mathtt{sync}_{\{x,y\}\to\{z,wt\}}?} s_j \xrightarrow{m^t?} s_j^{mid} \xrightarrow{m^t!} s_{bt} \in Path(CD_{\{z,wt\}})$, $|s_{kt}| \leq 1$

(iii.b) $\forall t \in [1..q]$: $s_j^{sync} \xrightarrow{\mathtt{sync}_{\{x,y\}\to\{z,wt\}}?} s_j \in Path(CD_{\{z,wt\}})$, $|s_{kt}| \geq 2$

if $\exists s_i^C, s_j^C, s_{k1}^C, s_{k2}^C, \ldots, s_{kq}^C \in \mathcal{S}'$: $|s_j^C| \geq 2$,

$$s_i^C \xrightarrow{m_{P_x \to P_y}} s_j^C \ , \ s_j^C \xrightarrow{m^1_{P_z \to P_{w1}}} s_{k1}^C \ , \ s_j^C \xrightarrow{m^2_{P_z \to P_{w2}}} s_{k2}^C \ , \ \ldots \ , \ s_j^C \xrightarrow{m^q_{P_z \to P_{wq}}} s_{kq}^C \ \textbf{and}$$
$$P_x \neq P_z$$

2. Coordination of independent sequences across independent branches

(i) $CD_{\{z,w\}} \in \mathcal{CD}^C, \quad \forall t \in [1..q]$: $CD_{\{xt,yt\}} \in \mathcal{CD}^C$,

(ii.a) $s_j \xrightarrow{m'?} s_j^{mid} \xrightarrow{m'!} s_k^{sync} \xrightarrow{\texttt{sync}_{\{z,w\} \to \{x1,y1\}\{x2,y2\}\ldots\{xq,yq\}}!} s_k^{branch} \in Path(CD_{\{z,w\}})$,

$\{z, w\} \notin \Big\{\{x1, y1\}, \{x2, y2\}, \ldots, \{xq, yq\}\Big\}$,

(ii.b) $\forall t \in [1..q]$: $s_k^{sync} \xrightarrow{\texttt{sync}_{\{z,w\} \to \{xt,yt\}}?} s_k^{branch} \in Path(CD_{\{xt,yt\}}), \ \{xt, yt\} \neq \{z, w\}$,

(iii) $\forall t \in [1..q]$: $s_k^{branch} \xrightarrow{\texttt{sync}_{\{xt,yt\} \to \{x1,y1\}\ldots\{xt-1,yt-1\}\{xt+1,yt+1\}\ldots\{xq,yq\}}!} s_k \xrightarrow{m^t?} s_k^{mid} \xrightarrow{m^t!} s_{bt} \in Path(CD_{\{xt,yt\}})$,

$$\forall h \in [1..t-1t+1..q]: \ s_k^{branch} \xrightarrow{\texttt{sync}_{\{xh,yh\} \to \{xt,yt\}}?} s_{bh} \in Path(CD_{\{xt,yt\}})$$

if $\exists s_j^C, s_k^C, s_{b1}^C, s_{b2}^C, \ldots, s_{bq}^C \in \mathcal{S}'$: $|s_j^C| \geq 2$,

$$s_j^C \xrightarrow{m'_{P_z \to P_w}} s_k^C \ , \quad s_k^C \xrightarrow{m^1_{P_{x1} \to P_{y1}}} s_{b1}^C \ , \quad s_k^C \xrightarrow{m^2_{P_{x2} \to P_{y2}}} s_{b2}^C \ , \ \ldots \ , \ s_k^C \xrightarrow{m^q_{P_{xq} \to P_{yq}}} s_{bq}^C \ \textbf{and}$$
$$\exists t, r \in [1..q]: \ P_{xt} \neq P_{xr} \neq P_z$$

Rule 1 is for synthesizing the additional synchronization messages and the related additional states (with respect to the states already inherited from E_C) that are required to handle a sequence of independent interactions across dependent branches. Indeed, Rule 1 is a generalization of the base case that accounts for handling a sequence of independent interactions of the form $s_i^C \xrightarrow{m_{P_x \to P_y}} s_j^C \xrightarrow{m'_{P_z \to P_w}} s_k^C$, with s_j^C being a plain state in the choreography EfM E_C with only one outgoing transition (see condition $|s_j^C| = 1$). The rule for the base case is reported below and can be easily achieved by setting $q = 1$, hence obtaining $|s_j^C| = 1$:

1. Coordination of independent sequences – base case

(i) $CD_{\{x,y\}} \in \mathcal{CD}^C, \ \ CD_{\{z,w\}} \in \mathcal{CD}^C$

(ii) $s_i \xrightarrow{m?} s_i^{mid} \xrightarrow{m!} s_j \xrightarrow{\texttt{sync}_{\{x,y\} \to \{z,w\}}!} s_j^{sync} \in Path(CD_{\{x,y\}})$

(iii.a) $s_j^{sync} \xrightarrow{\texttt{sync}_{\{x,y\} \to \{z,w\}}?} s_j \xrightarrow{m'?} s_j^{mid} \xrightarrow{m'!} s_k \in Path(CD_{\{z,w\}}), \ |s_k| \leq 1$

(iii.b) $s_j^{sync} \xrightarrow{\texttt{sync}_{\{x,y\} \to \{z,w\}}?} s_j \in Path(CD_{\{z,w\}}), \ |s_k| \geq 2$

if $\exists s_i^C, s_j^C, s_k^C \in \mathcal{S}'$: s_j^C is a plain state, $s_i^C \xrightarrow{m_{P_x \to P_y}} s_j^C \xrightarrow{m'_{P_z \to P_w}} s_k^C$ **and** $P_x \neq P_z$

The interactions $\xrightarrow{m_{P_x \to P_y}}$ and $\xrightarrow{m'_{P_z \to P_w}}$ are independent since the sender participants are different (see condition $P_x \neq P_z$). The CDs $CD_{\{x,y\}}$ and $CD_{\{z,w\}}$ are then needed (see assertion *1.i*). In order to enforce realizability, the two CDs synchronize (rendezvous) "across" the newly added state s_j^{sync} through the `sync` message, which is exchanged synchronously according to the assertions *1.ii* and *1.iii.a*. The assertion *1.iii.b* manages the situation when the state s_k^C is a branch state (i.e., $|s_k| \geq 2$). The only difference with respect to the assertion *1.iii.a*, is that only the `sync` message is "added" to $CD_{\{z,w\}}$; the handling of the message m' will be managed when Rule 2 will be applied to the branch state s_k^C as explained below.

By considering the portion of the choreography EfM that contains only the flows leading to the fork state s_6^C, Fig. 11 shows the (preliminary) transitions and states for $CD_{\{1,3\}}$ and $CD_{\{2,3\}}$ resulting from the application of the rule 1 to the path $s_0^C \xrightarrow{m1_{P1 \to P3}} s_1^C \xrightarrow{m2_{P2 \to P3}} s_2^C$.

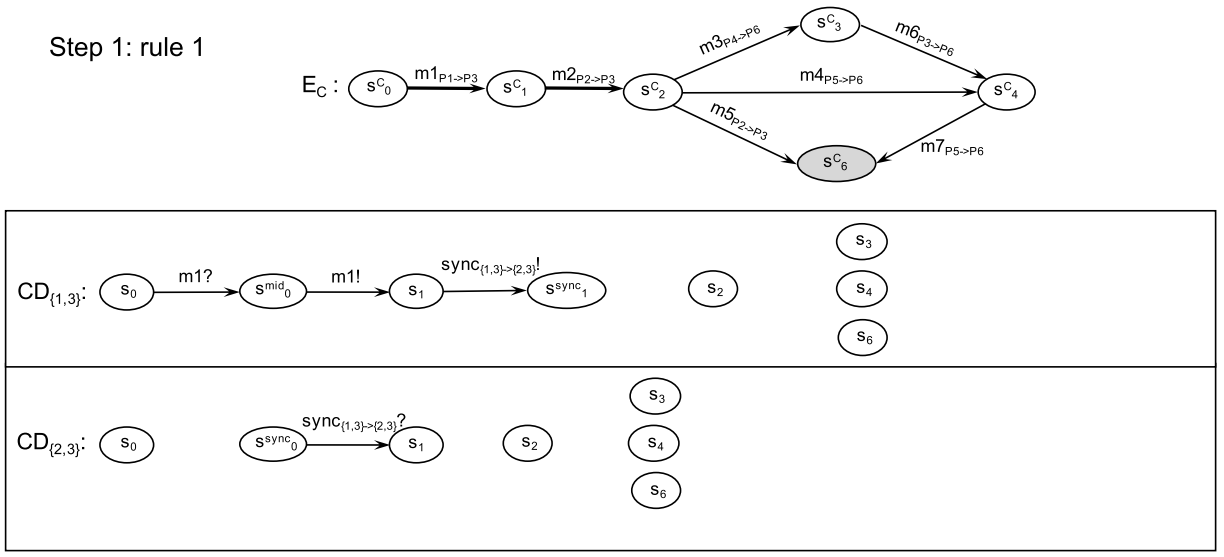

Fig. 11. Rule 1 – Coordination of independent sequences – base case.

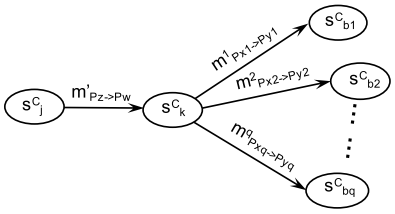

Fig. 12. Independent sequences across independent branches handled by Rule 2 – the general case.

Rule 2 is for synthesizing the additional synchronization messages and the related additional states that are required to handle independent interactions of the form shown in Fig. 12, where s_k^C is a branch state in the choreography EfM E_C with more than one outgoing transition (see condition $|s_k^C| \geq 2$). The state s_k^C identifies independent branches since at least two different sender participants, following different paths, are involved (see condition $\exists t, r \in [1..q]: P_{xt} \neq P_{xr} \neq P_z$). Moreover, the situation in the figure also accounts for independent sequences across the branch state s_k^C – see all the paths made of two transitions with the first transition being $s_j^C \xrightarrow{m'_{P_z \to P_w}} s_k^C$. For instance, for $q = 2$, one such a path is $s_j^C \xrightarrow{m'_{P_z \to P_w}} s_k^C \xrightarrow{m^2_{P_{x2} \to P_{y2}}} s_{b2}^C$. Thus, $CD_{\{z,w\}}$ is needed (see assertion *2.i*) to coordinate the transition entering the branching with all the outgoing transitions (see assertions *2.ii.a* and *2.ii.b*).

Then, for what concerns the transitions outgoing from s_k^C, a set of CDs is needed for enforcing mutual exclusion among the sending actions that can be fired from the branch state s_k^C by all the sender participants. Analogously to what is done for Rule 1, at the level of the CD abstract behavior, this is done through the synchronous exchange of `sync` messages. According to the assertion *2.iii*, each $CD_{\{xt,yt\}}$ that supervises a participant P_{xt} involved in a branch, either sends a `sync` message to all the other CDs involved in the same branch to be granted the permission to proceed, or can receive a `sync` message from any other competing CD, hence waiting until P_{xt} is allowed to proceed.

Fig. 13 shows the transitions and states for $CD_{\{1,3\}}$, $CD_{\{2,3\}}$, $CD_{\{4,6\}}$ and $CD_{\{5,6\}}$ resulting from the application of the Rule 2 across the branch state s_2^C. Next, Fig. 14 shows the resulting CDs after two subsequent applications of Rule 1 to the paths $s_2^C \xrightarrow{m3_{P4 \to P6}} s_3^C \xrightarrow{m6_{P3 \to P6}} s_4^C$ and $s_3^C \xrightarrow{m6_{P3 \to P6}} s_4^C \xrightarrow{m7_{P5 \to P6}} s_3^C$. Finally, Fig. 15 shows the complete abstract behavior models of the CDs obtained by suitably adding ϵ-transitions. This step is straightforward and, hence, we omit a formalization of it. Informally, ϵ-transitions among disconnected inherited states are added according to the choreography specification, similarly to what is done for choreography projection (Definition 3). For what concerns adding an ϵ-transition between an inherited state, different from s_0, and a *sync* state s_i^{sync}, if s_i^{sync} has no incoming (resp., outgoing) transitions, then an ϵ-transition is added from s_{i-1} (resp., s_i^{sync}) to s_i^{sync} (resp., s_{i+1}). Furthermore, if s_0 has no outgoing transitions, then an ϵ-transition from s_0 to s_0^{sync} is added.

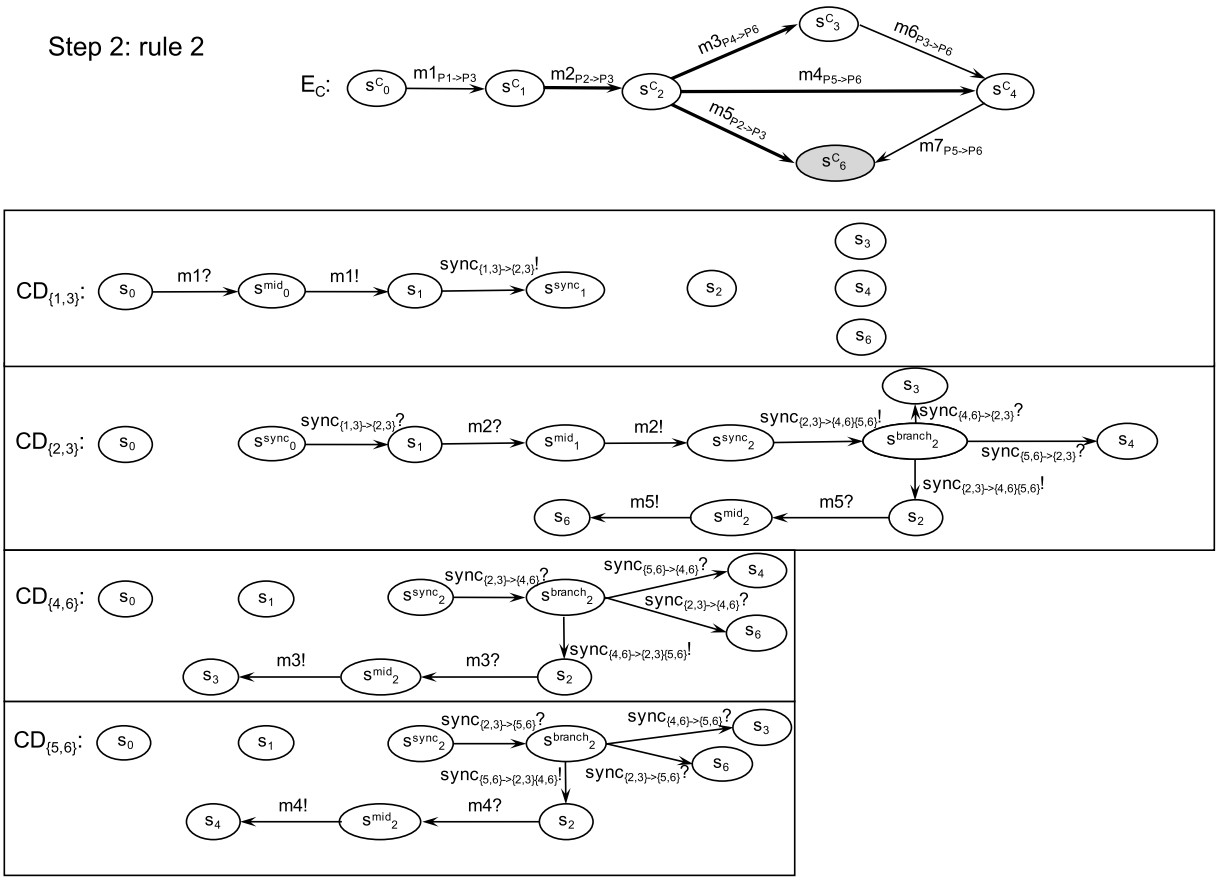

Fig. 13. Rule 2 – Coordination of independent sequences across independent branches.

3.4.2. Dealing with parallel flows

Definition 7 generalizes Definition 6 by also taking into account simultaneous actions. This means dealing with choreography transitions labeled with sets of message exchanges in addition to transitions labeled with single message exchanges. As a result, the generated CDs will be able to deal with simultaneous synchronization actions (simultaneous `sync` actions) and will support the highest possible degree of real parallelism during the choreography execution. In order to facilitate the reading and the comprehension of the CDs synthesis method in case of simultaneous actions, in Definition 7 we make use of the following shortcuts.

Given a set of simultaneous message exchanges M, CD^M is the set of indexes of the CDs involved in the simultaneous message exchanges represented by M. For instance, considering $s_6^C \xrightarrow{\{m7_{P3\to P1}, m9_{P2\to P3}\}} (s_7^C, s_8^C)$, we can write $s_6^C \xrightarrow{M} (s_7^C, s_8^C)$, where $M = \{m7_{P3\to P1}, m9_{P2\to P3}\}$, and $\mathtt{CD}^M = \{\{3,1\},\{2,3\}\}$.

When used to label `sync` actions, we will make use of the following shortcuts: CD^M denotes the concatenation of all the indexes of CDs involved in M. Given the CD indexes $\{x, y\}$, $CD^M \setminus \{x, y\}$ is the above concatenation of indexes except for $\{x, y\}$. Similarly, $\mathtt{CD}^{M^1}\mathtt{CD}^{M^2}\cdots\mathtt{CD}^{M^q}$ concatenates the indexes of all the CDs involved in $M^1, M^2, \ldots, M^q$.

Given a set of simultaneous message exchanges M, we also define $M?_{\{z,w\}} = \{m? \mid m_{P_z\to P_w} \in M\}$, and $M!_{\{z,w\}} = \{m! \mid m_{P_z\to P_w} \in M\}$. For instance, given $M = \{m7_{P3\to P1}, m9_{P2\to P3}, m6_{P3\to P1}\}$, $\mathtt{M?_{\{3,1\}}} = \{m7?, m6?\}$ and $\mathtt{M!_{\{3,1\}}} = \{m7!, m6!\}$. Given a set of simultaneous messages involving more than one pair of participants and the indexes of a specific CD, this shortcut serves to extract the messages that must be coordinated by the specific CD, and to easily create the two related input and output sets that permit delegation through decoupling.

Definition 7 *(CDs Abstract Behavior Synthesis).* Let $C = (\mathcal{P}, \mathcal{S}^C, s_0^C, \mathcal{L}, \mathcal{T}^C)$ be a choreography specification where $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ is the set of participants. Let $E_C = (\mathcal{P}, \mathcal{S}', s_0^C, \mathcal{L}, \mathcal{T}', \mathcal{S}_f^F)$ be the EfM of C. The set $\mathcal{CD}^C \subseteq \left\{CD_{\{i,j\}} \mid i, j \in [1..n]\right\}$ of the CD abstract behaviors that are needed to enforce the realization of C is such that for all $CD_{\{x,y\}} = (\mathcal{V}_{\{x,y\}}, s_0, \mathcal{M}_{\{x,y\}}, \delta_{\{x,y\}}) \in \mathcal{CD}^C$ we have that $\forall s_i^C \in \mathcal{S}'$: $s_i \in \mathcal{V}_{\{x,y\}}$ (i.e., each CD inherits its states from C) and:

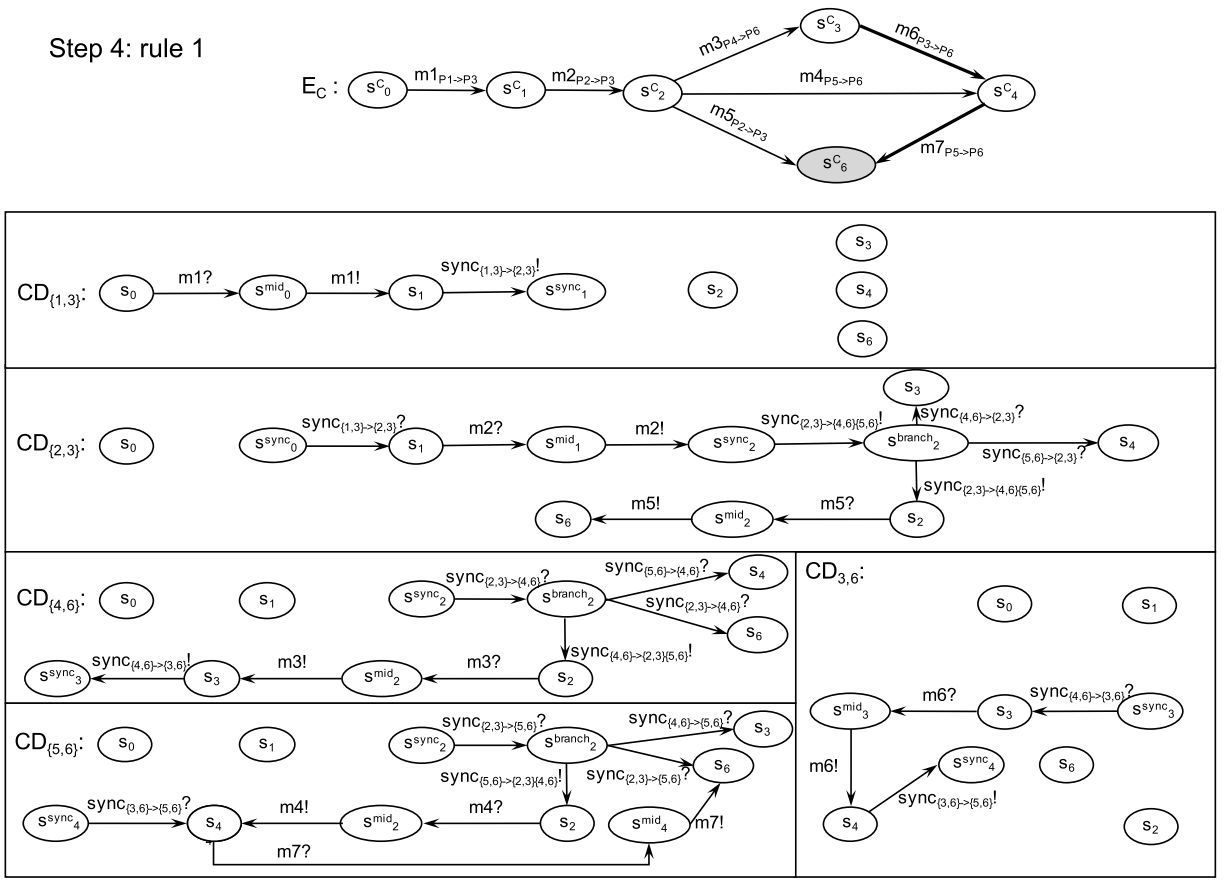

Fig. 14. Rule 1 – Coordination of independent sequences – base case.

1. Coordination of independent sequences across dependent branches/parallel flows

$$(i)\ \forall\{x, y\} \in \mathrm{CD}^{\mathtt{M}'} : CD_{\{x,y\}} \in \mathcal{CD}^{C},$$

$$\forall t \in [1..q]: \forall\{z, w\} \in \mathrm{CD}^{\mathtt{M}^t} : CD_{\{z,w\}} \in \mathcal{CD}^{C},$$

$$(ii)\ \forall\{x, y\} \in \mathrm{CD}^{\mathtt{M}'} :$$

$$s_j \xrightarrow{\mathtt{M}'?\{x,y\}} s_i^{mid} \xrightarrow{\mathtt{M}'!\{x,y\}} s_j \xrightarrow{\mathrm{sync}_{\{x,y\}\to \mathrm{CD}^{\mathtt{M}^1}\ \mathrm{CD}^{\mathtt{M}^2}\ldots\mathrm{CD}^{\mathtt{M}^q}\setminus\{x,y\}}!} s_j^{sync} \in Path(CD_{\{x,y\}}),$$

$$(iii.a)\ \forall t \in [1..q]: \forall\{z, w\} \in \mathrm{CD}^{\mathtt{M}^t} : s_j^{sync} \xrightarrow{\mathrm{sync}_{\mathrm{CD}^{\mathtt{M}'}\setminus\{z,w\}\to\{z,w\}}?} s_j \xrightarrow{\mathtt{M}^t?\{z,w\}} s_j^{mid} \xrightarrow{\mathtt{M}^t!\{z,w\}} \in Path(CD_{\{z,w\}}),$$

$$|s_{kt}| \leq 1$$

$$(iii.b)\ \forall t \in [1..q]: \forall\{z, w\} \in \mathrm{CD}^{\mathtt{M}^t} : s_j^{sync} \xrightarrow{\mathrm{sync}_{\mathrm{CD}^{\mathtt{M}'}\setminus\{z,w\}\to\{z,w\}}?} s_j \in Path(CD_{\{z,w\}}), |s_{kt}| \geq 2$$

if $\exists s_i^C, s_j^C, s_{k1}^C, s_{k2}^C, \ldots, s_{kq}^C \in \mathcal{S}' : |s_j^C| \geq 2,$

$$s_i^C \xrightarrow{\mathtt{M}'} s_j^C\ ,\quad s_j^C \xrightarrow{\mathtt{M}^1} s_{k1}^C\ ,\quad s_j^C \xrightarrow{\mathtt{M}^2} s_{k2}^C\ ,\quad \ldots\ ,\quad s_j^C \xrightarrow{\mathtt{M}^q} s_{kq}^C \quad \textbf{and}$$

$$\{x, y\} \in \mathrm{CD}^{\mathtt{M}'},\ \exists t \in [1..q]: \{z, w\} \in \mathrm{CD}^{\mathtt{M}^t} : P_x \neq P_z,$$

$$\forall t \in [1..q]: \forall m_{P\to P'} \in \mathtt{M}^t : P = P_z$$

2. Coordination of independent sequences across independent branches/parallel flows

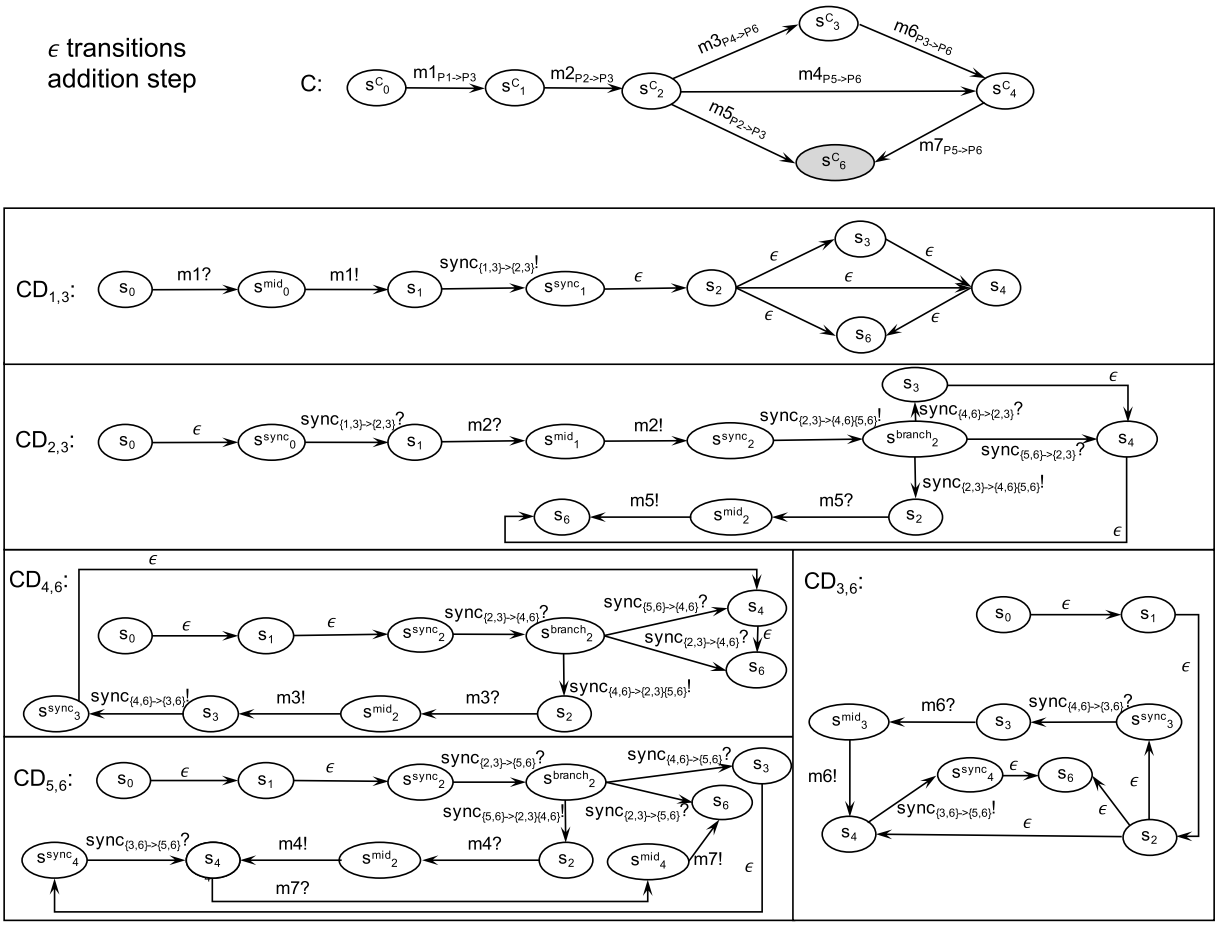

Fig. 15. Adding internal actions.

(i) $\forall\{z, w\} \in \mathrm{CD}^{\mathtt{M}'} : CD_{\{z,w\}} \in \mathcal{CD}^C$,

$\forall t \in [1..q]: \forall\{x, y\} \in \mathrm{CD}^{\mathtt{M}^t} : CD_{\{x,y\}} \in \mathcal{CD}^C$,

(ii.a) $\forall\{z, w\} \in \mathrm{CD}^{\mathtt{M}'}$:

$$s_j \xrightarrow{\mathtt{M}'?\{z,w\}} s_j^{mid} \xrightarrow{\mathtt{M}'!\{z,w\}} s_k^{sync} \xrightarrow{\mathrm{sync}_{\{z,w\}\to \mathrm{CD}^{\mathtt{M}^1}\,\mathrm{CD}^{\mathtt{M}^2}\ldots\mathrm{CD}^{\mathtt{M}^q}\setminus\{z,w\}}!} s_k^{branch} \in Path(CD_{\{z,w\}}),$$

(ii.b) $\forall t \in [1..q]: \forall\{x, y\} \in \mathrm{CD}^{\mathtt{M}^t} : s_k^{sync} \xrightarrow{\mathrm{sync}_{\mathrm{CD}^{\mathtt{M}'}\setminus\{x,y\}\to\{x,y\}}?} s_k^{branch} \in Path(CD_{\{x,y\}})$,

(iii) $\forall t \in [1..q] : \forall\{x, y\} \in \mathrm{CD}^{\mathtt{M}^t}$:

$$s_k^{branch} \xrightarrow{\mathrm{sync}_{\{x,y\}\to \mathrm{CD}^{\mathtt{M}^1}\ldots\mathrm{CD}^{\mathtt{M}^{t-1}}\,\mathrm{CD}^{\mathtt{M}^{t+1}}\ldots\mathrm{CD}^{\mathtt{M}^q}\setminus\{x,y\}}!} s_k \xrightarrow{\mathtt{M}^t?\{x,y\}} s_k^{mid} \xrightarrow{\mathtt{M}^t!\{x,y\}} s_{bt} \in Path(CD_{\{x,y\}}),\ |\mathtt{M}^t| = 1,$$

$$s_k^{branch} \xrightarrow{\mathrm{sync}_{\mathrm{CD}^{\mathtt{M}^t}\to\mathrm{CD}^{\mathtt{M}^t}}?!} s_k \xrightarrow{\mathtt{M}^t?\{x,y\}} s_k^{mid} \xrightarrow{\mathtt{M}^t!\{x,y\}} s_{bt} \in Path(CD_{\{x,y\}}),\ |\mathtt{M}^t| \geq 1,$$

$$\forall h \in [1..t-1t+1..q] : s_k^{branch} \xrightarrow{\mathrm{sync}_{\mathrm{CD}^{\mathtt{M}^h}\setminus\{x,y\}\to\{x,y\}}?} s_{bh} \in Path(CD_{\{x,y\}})$$

if $\exists s_j^C, s_k^C, s_{b1}^C, s_{b2}^C, \ldots, s_{bq}^C \in \mathcal{S}' : |s_k^C| \geq 2$,

$$s_j^C \xrightarrow{\mathtt{M}'} s_k^C\ ,\ s_k^C \xrightarrow{\mathtt{M}^1} s_{b1}^C\ ,\ s_k^C \xrightarrow{\mathtt{M}^2} s_{b2}^C\ ,\ \ldots\ ,\ s_k^C \xrightarrow{\mathtt{M}^q} s_{bq}^C \quad \textbf{and}$$

$\{z, w\} \in \mathrm{CD}^{\mathtt{M}'},\ \exists t, r \in [1..q]: \{xt, yt\} \in \mathrm{CD}^{\mathtt{M}^t},\ \{xr, yr\} \in \mathrm{CD}^{\mathtt{M}^r} : P_{xt} \neq P_{xr} \neq P_z$

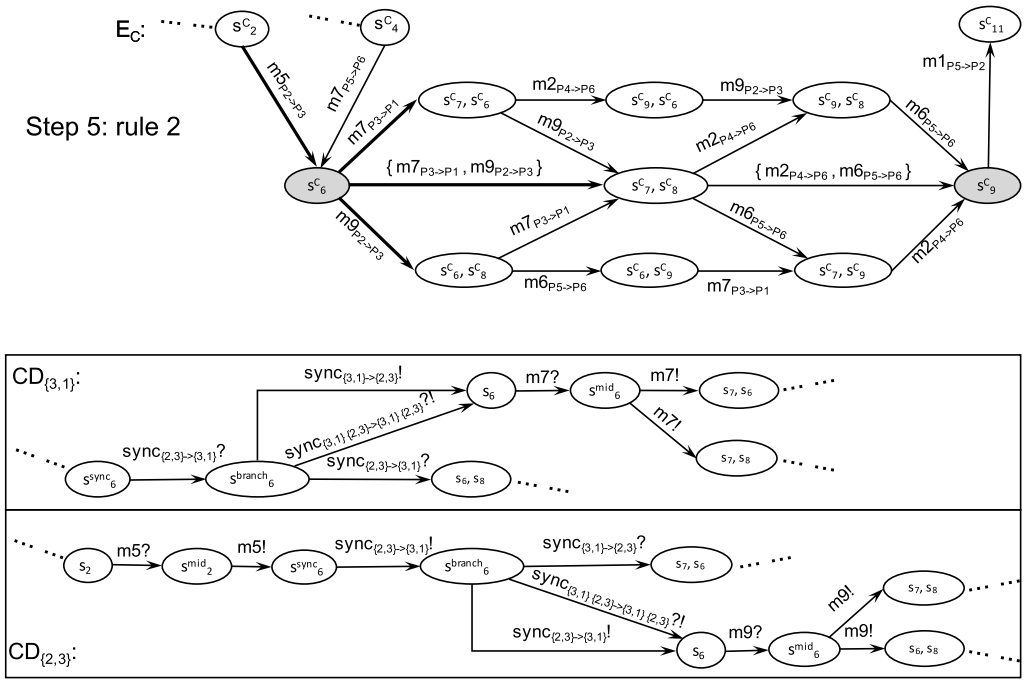

Fig. 16. Rule 2 (Definition 7) – Coordination of independent sequences across independent branches/parallel flows.

Rule 1 enhances the same rule of Definition 6 in order to synthesize the additional synchronization messages (possibly simultaneous) and the related additional states (with respect to the states already inherited from E_C) that are required to handle sequences of independent interactions across dependent branches that now can also concern the execution of simultaneous actions, i.e., branches in E_C originating from parallel flows of C.

Rule 2 is an enhancement of the same rule of Definition 6 and permit to handle sequences of independent interactions across independent branches that can also concern simultaneous actions. Thus, similarly to Rule 1, the enhancement of Rule 2 permit to also deal with parallel flows.

It is easy to see that these rules have the same structure of rule 1 and rule 2 in Definition 6 but, leveraging the fact the EfM models all the possible interactions concerning parallel flows, both interleaved and simultaneous, they deal with set of simultaneous actions and synchronization actions that enable the simultaneous exchange of synchronization information for both one-to-many and many-to-one interactions (and hence also many-to-many interactions).

Continuing our sample choreography sample, in Fig. 16 we consider the portion of the choreography EfM E_C starting from the states entering the state s_6^C (highlighted in light-gray to recall it is a fork state in the choreography specification C). In the figure, both $CD_{\{3,1\}}$ and $CD_{\{2,3\}}$ have the many-to-many synchronization transition $s_6^{branch} \xrightarrow{synch_{\{3,1\},\{2,3\}\to\{3,1\},\{2,3\}}?!} s_6$ that permits the participants $P3$ and $P2$ to exchange the message $m7$ and $m9$ simultaneously, respectively. Note that, abusing notation in rule 2, the transition $s_6^{branch} \xrightarrow{synch_{\{3,1\},\{2,3\}\to\{3,1\},\{2,3\}}?!} s_6$ actually identifies the two transitions $s_6^{branch} \xrightarrow{synch_{\{3,1\},\{2,3\}\to\{3,1\},\{2,3\}}?} s_6$ and $s_6^{branch} \xrightarrow{synch_{\{3,1\},\{2,3\}\to\{3,1\},\{2,3\}}!} s_6$ meaning that it does not matter whether $CD_{\{3,1\}}$ sends the synchronization message and $CD_{\{2,3\}}$ receives it, or vice versa. Without loss of generality and without reducing the parallelism degree, we could have equivalently managed this situation by assigning a priority to all CDs so that, in the case of simultaneous actions, only the involved CD with the highest priority would have send the synchronization message and all the others would have received it.

Another observation is that in order to correctly produce the behavior of the CDs, during the synthesis process we admit the presence of non-deterministic transitions. At the end, we determinize all the CDs so that when building the model of the choreographed system we deal with all deterministic behaviors.

3.5. Choreographed system

In this section we formalize the behavior of a choreographed system obtained by composing in parallel a set of participants, each of them modeled by its concrete behavior, with a set of CDs as additional participants performing exogenous coordination, each of them modeled by its concrete behavior as well.

We recall that the synthesis process formalized by Definition 7 produces, for each CD to be synthesized, its abstract behavior. In the following definition, when defining the behavior of a choreographed system, we consider CDs modeled by their

concrete behavior. Referring to Definition 7, the concrete behavior of a CD abstract behavior $CD_{\{i,j\}} = (\mathcal{V}_{\{i,j\}}, s_0, \mathcal{M}_{\{i,j\}}, \delta_{\{i,j\}})$ is obtained by setting the type of all the ϵ-transitions to internal (ϵ), and the type of all the `sync` transitions to synchronous ($\bullet$). Then, let $B_i^C = (\mathcal{V}_i, v_0^i, \mathcal{M}_i, \delta_i^C, \mathcal{V}_{iJ}^F)$ and $B_j^C = (\mathcal{V}_j, v_0^j, \mathcal{M}_j, \delta_j^C, \mathcal{V}_{jJ}^F)$ be the concrete behaviors of P_i and P_j, respectively, for all $(v, M, v', type) \in \delta_i^C$ with $type \in \{\circ, \bullet\}$, the type of $(v, \overline{M}, v') \in \delta_{\{i,j\}}$ is set to $type$ where $\overline{M} = \{m? \mid m! \in M\} \cup \{m! \mid m? \in M\}$.

Definition 8 *(Choreographed System Behavior).* Let $\mathcal{P} = \{P_1^C, P_2^C, \ldots, P_n^C\}$ be a set of concrete participants selected from the inventory to realize a choreography specification C with abstract participants $P_1, P_2, \ldots, P_n$ whose roles are played by $P_1^C, P_2^C, \ldots, P_n^C$, respectively. Let $q_1, \ldots, q_n$ be the maximum size values predefined for their message queues $Q_1, \ldots, Q_n$ respectively, and let $B_i^C = (\mathcal{V}_i, v_{0i}, \mathcal{M}_i, \delta_i^C, \mathcal{V}_{iJ}^F)$ be the concrete behavior of P_i^C where $\mathcal{M}_i = \mathcal{M}_i^! \cup \mathcal{M}_i^?$ and $\forall i, j : i \neq j \implies \mathcal{M}_i^? \cap \mathcal{M}_j^? = \mathcal{M}_i^! \cap \mathcal{M}_j^! = \emptyset$.

Let $\mathcal{CD}^C = \left\{ CD_x = (\mathcal{W}_x, s_0^x, \mathcal{L}_x = \mathcal{L}_x^! \cup \mathcal{L}_x^?, \rho_x) \mid x \in [1..p] \right\}$ be the set of CD concrete behaviors that have been synthesized to enforce the realization of C, and let $B_1, \ldots, B_p$ be their 1-bounded message queues.

The behavior of a choreographed system composing the participants in $\mathcal{P}$ and the CDs in $\mathcal{CD}^C$ is modeled as a hybrid system behavior $CS = (\mathcal{P} \cup \mathcal{CD}^C, \mathcal{S}, s_0, \mathcal{M}, \Delta)$ extended to suitably deal with the interaction CDs–CDs and participants–CDs.

The interaction participants–participants that do not require coordination and the execution of internal actions are handled by inheriting **all the rules of Definition 5**. Concerning the interactions participants–CDs and CDs–CDs, for $s = (v_1, Q_1, \ldots, v_n, Q_n, w_1, B_1, \ldots, w_p, B_p) \in \mathcal{S}$ and $s' = (v_1', Q_1', \ldots, v_n', Q_n', w_1', B_1', \ldots, w_p', B_p') \in \mathcal{S}$, the transition relation $\Delta \subseteq \mathcal{S} \times 2^{(\mathcal{P} \times \mathcal{M} \times \mathcal{P}) \cup \{\epsilon\}} \setminus \emptyset \times \mathcal{S} \times \{\bullet, \circ, \epsilon\}$ is defined as follows:

Participants–CDs asynchronous actions (where $|Q_r|$ is the length of Q_r, $Q_r^0 = Q_r$, and $Q_r^l = Q_r'$, and the same for B_r)

$$
\begin{aligned}
&s \xrightarrow{\{x_1,\ldots,x_l\}}_{\circ} s' \in \Delta \text{ if}\\
&\exists i_1, \ldots, i_l \in [1..n],\ j_1, \ldots, j_l \in [1..p] :\ \exists m^1 \in \mathcal{M}_{i_1}^! \cap \mathcal{L}_{j_1}^?, \ldots, m^l \in \mathcal{M}_{i_l}^! \cap \mathcal{L}_{j_l}^?,\\
&m'^1 \in \mathcal{L}_{j_1}^! \cap \mathcal{M}_{i_1}^?, \ldots, m'^l \in \mathcal{L}_{j_l}^! \cap \mathcal{M}_{i_l}^? :\\
&v_{i_1} \xrightarrow{I_{i_1}^? \cup I_{i_1}^!}_{\circ} v_{i_1}' \in \delta_{i_1}^C, \ldots, v_{i_l} \xrightarrow{I_{i_l}^? \cup I_{i_l}^!}_{\circ} v_{i_l}' \in \delta_{i_l}^C,\\
&w_{j_1} \xrightarrow{I_{j_1}^? \cup I_{j_1}^!}_{\circ} v_{j_1}' \in \rho_{j_1}, \ldots, v_{j_l} \xrightarrow{I_{j_l}^? \cup I_{j_l}^!}_{\circ} v_{j_l}' \in \rho_{j_l},\\
&I_{i_1}^!, \ldots, I_{i_l}^! \in 2^{\{m^1!,\ldots,m^l!\}},\ I_{j_1}^?, \ldots, I_{j_l}^? \in 2^{\{m^1?,\ldots,m^l?\}},\\
&I_{j_1}^!, \ldots, I_{j_l}^! \in 2^{\{m'^1!,\ldots,m'^l!\}},\ I_{i_1}^?, \ldots, I_{i_l}^? \in 2^{\{m'^1?,\ldots,m'^l?\}},\ I_{i_1}^! \cap I_{i_1}^? = \emptyset, \ldots, I_{j_l}^! \cap I_{j_l}^? = \emptyset,\\
&\forall s, t \in [1..l] : m^s? \in \textstyle\bigcup_{h=1}^{l}(I_{i_h}^? \cup I_{j_h}^?), m^t! \in \bigcup_{h=1}^{l}(I_{i_h}^! \cup I_{j_h}^!) \implies Q_{j_s}^{s-1} = m^s Q_{j_s}^s,\ |B_s| = 0,\ B_s' = m^s,\ x_s = \epsilon_s,\\
&|Q_{j_t}^{t-1}| < q_{j_t},\ Q_{j_t}^t = Q_{j_t}^{t-1} m^t,\ B_t' = B_t,\ x_t = m^t_{P_{i_t} \to P_{j_t}},\\
&\forall k \in [1..n] : \forall t \in [1..l] : k \neq i_t, j_t \implies Q_k' = Q_k,\ v_k' = v_k, \text{ and}\\
&\forall k \in [1..n] : \forall t \in [1..l] : k \neq j_t \implies Q_k' = Q_k,\\
&\forall h \in [1..m] : \forall t \in [1..l] : h \neq i_t, j_t \implies B_h' = B_h,\ w_h' = w_h, \text{ and}\\
&\forall h \in [1..m] : \forall t \in [1..l] : h \neq j_t \implies B_h' = B_h
\end{aligned}
$$

Participants–CDs synchronous actions

$$
\begin{aligned}
&s \xrightarrow{\{x_1,\ldots,x_l\}}_{\bullet} s' \in \Delta \text{ if}\\
&\exists i_1, \ldots, i_l \in [1..n],\ j_1, \ldots, j_l \in [1..p] :\ \exists m^1 \in \mathcal{M}_{i_1}^! \cap \mathcal{L}_{j_1}^?, \ldots, m^l \in \mathcal{M}_{i_l}^! \cap \mathcal{L}_{j_l}^?,\\
&m'^1 \in \mathcal{L}_{j_1}^! \cap \mathcal{M}_{i_1}^?, \ldots, m'^l \in \mathcal{L}_{j_l}^! \cap \mathcal{M}_{i_l}^? :\\
&v_{i_1} \xrightarrow{I_{i_1}^? \cup I_{i_1}^!}_{\bullet} v_{i_1}' \in \delta_{i_1}^C, \ldots, v_{i_l} \xrightarrow{I_{i_l}^? \cup I_{i_l}^!}_{\bullet} v_{i_l}' \in \delta_{i_l}^C,\\
&w_{j_1} \xrightarrow{I_{j_1}^? \cup I_{j_1}^!}_{\bullet} v_{j_1}' \in \rho_{j_1}, \ldots, v_{j_l} \xrightarrow{I_{j_l}^? \cup I_{j_l}^!}_{\bullet} v_{j_l}' \in \rho_{j_l},\\
&I_{i_1}^!, \ldots, I_{i_l}^! \in 2^{\{m^1!,\ldots,m^l!\}},\ I_{j_1}^?, \ldots, I_{j_l}^? \in 2^{\{m^1?,\ldots,m^l?\}},\\
&I_{j_1}^!, \ldots, I_{j_l}^! \in 2^{\{m'^1!,\ldots,m'^l!\}},\ I_{i_1}^?, \ldots, I_{i_l}^? \in 2^{\{m'^1?,\ldots,m'^l?\}},\ I_{i_1}^! \cap I_{i_1}^? = \emptyset, \ldots, I_{j_l}^! \cap I_{j_l}^? = \emptyset,\\
&\forall s, t \in [1..l] : m^s? \in \textstyle\bigcup_{h=1}^{l}(I_{i_h}^? \cup I_{j_h}^?), m^t! \in \bigcup_{h=1}^{l}(I_{i_h}^! \cup I_{j_h}^!) \implies Q_{j_s}^{s-1} = m^s Q_{j_s}^s,\ |B_s| = 0,\ B_s' = m^s,\ x_s = \epsilon_s,\\
&|Q_{j_t}^{t-1}| < q_{j_t},\ Q_{j_t}^t = Q_{j_t}^{t-1} m^t,\ B_t' = B_t,\ x_t = m^t_{P_{i_t} \to P_{j_t}},\\
&\forall t \in [1..l] : m^t? \in \textstyle\bigcup_{h=1}^{l}(I_{i_h}^? \cup I_{j_h}^?), m^t! \in \bigcup_{h=1}^{l}(I_{i_h}^! \cup I_{j_h}^!) \implies x_t = m^t_{P_{i_t} \to P_{j_t}},\\
&\forall k \in [1..n] : Q_k' = Q_k \text{ and } \forall k \in [1..n] : \forall t \in [1..l] : k \neq i_t, j_t \implies v_k' = v_k,\\
&\forall h \in [1..m] : B_h' = B_h \text{ and } \forall h \in [1..m] : \forall t \in [1..l] : h \neq i_t, j_t \implies w_h' = w_h
\end{aligned}
$$

CDs–CDs additional communication

$s \xrightarrow{\epsilon} s' \in \Delta$ if $\exists q1, q2 \in [1..p] : \exists x_1, \ldots, x_{q1}, y_1, \ldots, y_{q2} \in [1..p]$:
$\exists z_1, \ldots, z_{q1}, w_1, \ldots w_{q1}, h_1, \ldots, h_{q2}, k_1, \ldots, k_{q2} \in [1..n]$:
$\forall i \in [1..q] : CD_{\{z_i, w_i\}} = CD_{x_i} \in \mathcal{CD}^C$, $\texttt{sync}_{\{z_1,w_1\}\{z_2,w_2\}\ldots\{z_{q1},w_{q1}\}\rightarrow\{h_1,k_1\}\{h_2,k_2\}\ldots\{h_{q2},k_{q2}\}} \in \mathcal{L}^!_{x_i}$,
$w_{x_i} \xrightarrow{\texttt{sync}_{\{z_1,w_1\}\{z_2,w_2\}\ldots\{z_{q1},w_{q1}\}\rightarrow\{h_1,k_1\}\{h_2,k_2\}\ldots\{h_{q2},k_{q2}\}}!} \bullet\, w'_{x_i} \in \rho_{x_i}$,
$\forall t \in [1..p] : CD_{yt} = CD_{\{ht,kt\}}$, $\texttt{sync}_{\{z_1,w_1\}\{z_2,w_2\}\ldots\{z_{q1},w_{q1}\}\rightarrow\{h_1,k_1\}\{h_2,k_2\}\ldots\{h_{q2},k_{q2}\}} \in \mathcal{L}^?_{yt}$,
$w_{yt} \xrightarrow{\texttt{sync}_{\{z_1,w_1\}\{z_2,w_2\}\ldots\{z_{q1},w_{q1}\}\rightarrow\{h_1,k_1\}\{h_2,k_2\}\ldots\{h_{q2},k_{q2}\}}?} \bullet\, w'_{yt} \in \rho_{yt}$,
$\forall u \in [1..n] : Q'_u = Q_u,\ v'_u = v_u$,
$\forall r \in [1..m] : B'_r = B_r$ and $\forall r \in [1..m] : r \neq x_1, \ldots, x_q, y_1, \ldots, y_q \implies w'_r = w_r$

The intuition behind the rules in Definition 8 is that, only when needed, the CDs decouple the interactions among the participants and coordinate them according to the specified choreography. It is worth to recall that the decoupling is realized via 1-bounded message queues. Thus, at the level of the choreographed system, we are interested on observing the events in which a CD forwards previously received messages to the designated receiving participants. In fact, it is in that moment that the message exchanges specified in the choreography has been accomplished, independently from when the messages were received by the CD and, hence, from the order CDs receive actions are performed in the system. In other words, in order to check language equivalence between the choreographed system and the choreography specification, it is important for the choreographed system to exhibit the message exchanges accomplished through CDs send actions (possibly simultaneous). In this direction, note also that the exchange of synchronization actions produces an internal action at the level of the choreographed system.

For instance, considering the portion of the choreography EfM shown in Fig. 15, let us suppose that two concrete participants for $P1$ and $P2$ have been selected such that, from the initial states of their respective concrete behaviors $B1^C$ and $B2^C$, $B1^C$ asynchronously performs $m1!$ and $B2^C$ asynchronously performs $m2!$. Disregarding, for the sake of clarity, the rest of the choreography and the behaviors of the other selected participants, this means that from the choreographed system initial state the following two traces[10] are obtained by applying Definition 8 (beyond other traces):

$t1 = \epsilon_1\, \epsilon_2\, m1_{P1\rightarrow P3}\, \epsilon_3\, \epsilon_4\, \epsilon_5\, m2_{P2\rightarrow P3}$;
$t2 = \epsilon_4\, \epsilon_1\, \epsilon_2\, m1_{P1\rightarrow P3}\, \epsilon_3\, \epsilon_5\, m2_{P2\rightarrow P3}$.

The paths corresponding to $t1$ and $t2$ reach the same state, i.e., they are confluent. Note that $t1$ and $t2$ are equivalent under the notion of language equivalence that we introduced above. In particular, ϵ_1 corresponds to the asynchronous sending by $P1$ of the message $m1$ to $CD_{\{1,3\}}$. That is, $m1$ is stored into the queue of $CD_{\{1,3\}}$, hence making it full. At this point, subsequent send actions will be blocking until $m1$ is consumed. ϵ_2 corresponds to the consumption of $m1$ by $CD_{\{1,3\}}$. ϵ_3 corresponds to $CD_{\{1,3\}}$ synchronizing with $CD_{\{2,3\}}$ so to correctly coordinate each other. ϵ_4 corresponds to the asynchronous sending by $P2$ of $m2$ to $CD_{\{2,3\}}$. Finally, ϵ_5 corresponds to the consumption of $m2$ by $CD_{\{2,3\}}$.

By referring to ϵ_4 in $t2$, the exchange of $m2$ between $P2$ and $P3$ (i.e., $m2_{P2\rightarrow P3}$ from S_1^C to S_2^C in C) has not been accomplished yet. Thus, although $P2$ has already sent $m2$ to $CD_{\{2,3\}}$, this is still compliant to what is specified by C.

From the above example, we can notice that whatever the order in which CDs receive messages is, they are sent to the designed participants by respecting the order of the message exchanges specified by the choreography. The same holds when simultaneous actions due to parallel flows are executed.

3.6. Correctness

In this section, we show that the CDs synthesized by performing the method formalized in Section 3.4 are correct by construction. Informally, this requires showing that the language of the obtained choreographed system is equivalent to the one of the choreography EfM, under our notion of language equivalence. This essentially means that the introduction of the CDs in the system enforces the realizability of the specified choreography although it is in principle unrealizable.

As discussed above, the assumptions we make for achieving correctness-by-construction concern characteristics of the concrete participants selected from the inventory and the fact that we deal with well-specified choreographies. Leveraging the notion of projection, our method is able to check whether a choreography is well-specified or not. In any case, a well-specified choreography is always obtainable through message relabeling. Still exploiting projection, our method is also able to check if the EfM of the behavior of the concrete participants selected from the inventory is an alternating refinement of the one of the abstract participants in the choreography specification. If it is the case, the method proceeds by automatically synthesizing the correct-by-construction CDs that are needed to enforce the specified choreography.

The following theorem states correctness of the CDs synthesis method and gives a formal proof of it.

[10] ϵ-actions are indexed to ease the explanation.

Theorem 1 *(Correctness). Let* $C = (\mathcal{P}, \mathcal{S}^C, s_0^C, \mathcal{L}, \mathcal{T}^C)$ *be a (well-specified) choreography specification, where* $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ *is the set of participants, whose EfM is* $E_C = (\mathcal{P}, \mathcal{S}', s_0^C, \mathcal{L}, \mathcal{T}', \mathcal{S}_J^F)$*. Let* $\mathcal{P}^C = \{P_1^C, P_2^C, \ldots, P_n^C\}$ *be a set of concrete participants selected from the inventory to realize C (i.e.,* P_i^C *plays the role of* P_i*), let* $q_1, \ldots, q_n$ *be the maximum size values predefined for their message queues* $Q_1, \ldots, Q_n$ *respectively, and let* $B_i^C = (\mathcal{V}_i, v_{0i}, \mathcal{M}_i, \delta_i^C, \mathcal{V}_{iJ}^F)$ *be the EfM modeling the concrete behavior of* P_i^c*. Let* $\mathcal{CD}^C = \left\{CD_x = (\mathcal{W}_x, s_0^x, \mathcal{L}_x, \rho_x) \mid x \in [1..m]\right\}$ *be the set of EfMs, modeling the CD concrete behaviors, that have been synthesized to enforce the realization of C, and let* $B_1, \ldots, B_m$ *be their 1-bounded message queues. Let* $CS = (\mathcal{P} \cup \mathcal{CD}^C, \mathcal{S}, s_0, \mathcal{M}, \Delta)$ *be the EfM modeling the behavior of the choreographed system composing the participants in* $\mathcal{P}^c$ *and the CDs in* $\mathcal{CD}^C$*. If* $\forall i \in [1..n] : B_i^C \bowtie \Pi^C(P_i)$ *then* $CS \cong E_C$*.*

Proof. By contradiction, let us suppose that $\forall i \in [1..n] : B_i^C \bowtie \Pi^C(P_i)$ and that CS and E_C are not (language) equivalent. This means that (i) $\exists t_{cs} \in L(CS) : \nexists t_c \in L(E_C) : t_{cs} \cong t_c$ and (ii) $\exists t_c \in L(E_C) : \nexists t_{cs} \in L(CS) : t_{cs} \cong t_c$.

Case i – By the hypothesis, Definition 8, and the fact that we deal with participants and CDs whose behavior is deterministic, there exist two traces $t'_{cs} \in L(CS) \cup \{\epsilon\}$ and $t'_c \in L(E_C) \cup \{\epsilon\}$ such that $t'_{cs} \cong t'_c$, and:

1. $\exists M \in 2^{\mathcal{M} \cup \{\epsilon\}} : t_{cs} = t'_{cs}\, \epsilon^*\, M\, \epsilon^* \wedge \nexists t_c \in L(E_C) : t_c = t'_c\, M$
2. $\exists M, M' \in 2^{\mathcal{M} \cup \{\epsilon\}} : \exists m_{P_x \to P_y} \in M, m'_{P_z \to P_w} \in M' : x \neq z, t_{cs} = t'_{cs}\, \epsilon^*\, M\, \epsilon^*\, M'\, \epsilon^* \wedge \nexists t_c \in L(E_C) : t_c = t'_c\, M\, M'$
3. $\exists M, M'_1, M'_2, \ldots, M'_q \in 2^{\mathcal{M} \cup \{\epsilon\}} : \exists m_{P_x \to P_y} \in M, m'_{P_z \to P_{w1}} \in M'_1, m'_{P_z \to P_{w2}} \in M'_2, \ldots, m'_{P_z \to P_{wq}} \in M'_q : x \neq z,$
   $(t_{cs} = t'_{cs}\, \epsilon^*\, M\, \epsilon^*\, M'_1\, \epsilon^* \wedge \nexists t_c \in L(E_C) : t_c = t'_c\, M\, M'_1) \vee$
   $(t_{cs} = t'_{cs}\, \epsilon^*\, M\, \epsilon^*\, M'_2\, \epsilon^* \wedge \nexists t_c \in L(E_C) : t_c = t'_c\, M\, M'_2) \vee \ldots \vee$
   $(t_{cs} = t'_{cs}\, \epsilon^*\, M\, \epsilon^*\, M'_q\, \epsilon^* \wedge \nexists t_c \in L(E_C) : t_c = t'_c\, M\, M'_q)$
4. $\exists M, M'_1, M'_2, \ldots, M'_q \in 2^{\mathcal{M} \cup \{\epsilon\}} : \exists m_{P_z \to P_w} \in M, m'_{P_{x1} \to P_{y1}} \in M'_1, m'_{P_{x2} \to P_{y2}} \in M'_2, \ldots, m'_{P_{xq} \to P_{yq}} \in M'_q : \exists t, r \in [1..q] : xt \neq xr,$
   $(t_{cs} = t'_{cs}\, \epsilon^*\, M\, \epsilon^*\, M'_1\, \epsilon^* \wedge \nexists t_c \in L(E_C) : t_c = t'_c\, M\, M'_1) \vee$
   $(t_{cs} = t'_{cs}\, \epsilon^*\, M\, \epsilon^*\, M'_2\, \epsilon^* \wedge \nexists t_c \in L(E_C) : t_c = t'_c\, M\, M'_2) \vee \ldots \vee$
   $(t_{cs} = t'_{cs}\, \epsilon^*\, M\, \epsilon^*\, M'_q\, \epsilon^* \wedge \nexists t_c \in L(E_C) : t_c = t'_c\, M\, M'_q)$

Case i.1 invalidates the hypothesis, i.e., $\forall i \in [1..n] : B_i^C \bowtie \Pi^C(P_i)$. In fact, since the concrete participants are selected to be an alternating refinement of the abstract participants, $L(CS)$ could not have the trace t_{cs} without having the equivalent trace t_c in $L(E_C)$, otherwise the concrete participants P_k in CS, and involved in M, would not be an alternating refinement of the corresponding abstract participants P_k in E_C. Furthermore, **Case i.2** and **Case i.3** invalidate Rule 1 of Definition 7 in both the base case and the general one, respectively. In fact, by the assertions in Rule 1, the involved CDs ensure by construction that $t_{cs} \in L(CS)$. However, these CDs are synthesized only if $t_c \in L(E_C)$ (see the condition in Rule 1). Similarly, **Case i.4** invalidates Rule 2 of Definition 7 in that $t_{cs} \in L(CS)$ by the assertions of the rule. However, also in this case, by considering the traces corresponding to the paths used to express the condition of Rule 2, these assertions hold only if $t_c \in L(E_C)$. Thus, each case leads to a contradiction and, hence, the proof is given for case i.

Case ii – By the hypothesis, Definition 8, and the fact that we deal with participants and CDs whose behavior is deterministic, there exist two traces $t'_c \in L(E_C) \cup \{\epsilon\}$ and $t'_{cs} \in L(CS) \cup \{\epsilon\}$ such that $t'_c \cong t'_{cs}$, and:

1. $\exists M \in 2^{\mathcal{L} \cup \{\epsilon\}} : t_c = t'_c\, \epsilon^*\, M\, \epsilon^* \wedge \nexists t_{cs} \in L(CS) : t_{cs} = t'_{cs}\, M$
2. $\exists M, M' \in 2^{\mathcal{L} \cup \{\epsilon\}} : \exists m_{P_x \to P_y} \in M, m'_{P_z \to P_w} \in M' : x \neq z, t_c = t'_c\, \epsilon^*\, M\, \epsilon^*\, M'\, \epsilon^* \wedge \nexists t_{cs} \in L(CS) : t_{cs} = t'_{cs}\, M\, M'$
3. $\exists M, M'_1, M'_2, \ldots, M'_q \in 2^{\mathcal{L} \cup \{\epsilon\}} : \exists m_{P_x \to P_y} \in M, m'_{P_z \to P_{w1}} \in M'_1, m'_{P_z \to P_{w2}} \in M'_2, \ldots, m'_{P_z \to P_{wq}} \in M'_q : x \neq z,$
   $(t_c = t'_c\, \epsilon^*\, M\, \epsilon^*\, M'_1\, \epsilon^* \wedge \nexists t_{cs} \in L(CS) : t_{cs} = t'_{cs}\, M\, M'_1) \vee$
   $(t_c = t'_c\, \epsilon^*\, M\, \epsilon^*\, M'_2\, \epsilon^* \wedge \nexists t_{cs} \in L(CS) : t_{cs} = t'_{cs}\, M\, M'_2) \vee \ldots \vee$
   $(t_c = t'_c\, \epsilon^*\, M\, \epsilon^*\, M'_q\, \epsilon^* \wedge \nexists t_{cs} \in L(CS) : t_{cs} = t'_c\, M\, M'_q)$
4. $\exists M, M'_1, M'_2, \ldots, M'_q \in 2^{\mathcal{L} \cup \{\epsilon\}} : \exists m_{P_z \to P_w} \in M, m'_{P_{x1} \to P_{y1}} \in M'_1, m'_{P_{x2} \to P_{y2}} \in M'_2, \ldots, m'_{P_{xq} \to P_{yq}} \in M'_q : \exists t, r \in [1..q] : xt \neq xr,$
   $(t_c = t'_c\, \epsilon^*\, M\, \epsilon^*\, M'_1\, \epsilon^* \wedge \nexists t_{cs} \in L(CS) : t_{cs} = t'_{cs}\, M\, M'_1) \vee$
   $(t_c = t'_c\, \epsilon^*\, M\, \epsilon^*\, M'_2\, \epsilon^* \wedge \nexists t_{cs} \in L(CS) : t_{cs} = t'_{cs}\, M\, M'_2) \vee \ldots \vee$
   $(t_c = t'_c\, \epsilon^*\, M\, \epsilon^*\, M'_q\, \epsilon^* \wedge \nexists t_{cs} \in L(CS) : t_{cs} = t'_{cs}\, M\, M'_q)$

Case ii.1 directly invalidates the hypothesis, i.e., $\forall i \in [1..n] : B_i^C \bowtie \Pi^C(P_i)$, if the path ρ'_c of E_C corresponding to t'_c does not traverse a fork state without traversing also its paired join state. Otherwise, if ρ'_c traverses a fork state without traversing also its paired join state, case ii.1 would not represent a contradiction. However, by the hypothesis, there always exists at least one $M \in 2^{\mathcal{L} \cup \{\epsilon\}}$ such that $t_c = t'_c\, \epsilon^*\, M\, \epsilon^*$ and $t'_{cs}\, M \in L(CS)$, otherwise the concrete participants P_k in CS, and involved in M, would not be an alternating refinement of the corresponding abstract participants P_k in E_C, hence invalidating again the hypothesis. Similarly, **Case ii.2** and **Case ii.3** invalidate Rule 1 of Definition 7 in both the cases discussed for case ii.1 that concerns the existence of the path ρ'_c of E_C corresponding to t'_c. In fact, by the assertions in Rule 1, the involved CDs ensure

by construction that there always exist at least one M and one M' such that $t_{cs} \in L(CS)$. Analogously, **Case i.4** invalidates Rule 2 of Definition 7. Thus, each case leads to a contradiction and, hence, the proof is given also for case ii. □

4. Discussion

By referring to the handling of independent branches and independent parallel flows, in this section we briefly discuss some practical implications related to the actual implementation code of the synthesized CDs and their run-time.

As described in our previous work [13], each CD implements a mutual exclusion algorithm suitable for our purposes. In particular, the algorithm inherits the distributed mutual exclusion principle and leverages some foundational notions (such as happened-before relation, partial ordering, and time-stamps) of the algorithm in [18]. More precisely, the enforcement of mutual exclusive access to a single resource in [18] is scaled up to the enforcement of independent branches according to arbitrarily complex choreographies.

Furthermore, referring to the assertion 2.*iii* of Rule 2 and considering aspects related to the actual run-time execution of CDs, when a $CD_{\{xh,yh\}}$ is in s_{bh}, its queue can contain a message since P_{xh} may have independently sent it. Since we deal with the coordination of black-box participants from outside, to manage this situation, we need a notion of controllability that, e.g., through timeouts and/or fault messages, allows $CD_{xh,yh}$ to discard the received message by internally removing it from the queue and P_{xh} to rollback to the state s_k^{branch}. In general, this situation can happen on independent branches with asynchronous interactions. Note that when dealing with exogenous coordination of black-box participants, assuming controllability of send actions is the best one can do. If we would consider white-box participants, the issue can be solved by acting on the participant code, or changing the choreography specification. For instance, in [4,11], a choreography specification is repaired in order to make it realizable. When repaired, the choreography can be realized by generating the participants from scratch via choreography projection. Note that this is a different context from ours where, instead, we are focused on reusing third-party (black-box) participants to enforce the realizability of a choreography specification from outside without changing it. Details about the possible policies that a concrete participant could adopt, e.g., rollback and retry, deviating on alternative branches, etc. are out of scope of this paper. However, it is worth to mention that, leveraging Rule 2, our synthesis method can give useful feedback to the developer, hence supporting the implementation of the portion of the CD actual code that, in this case, cannot be generated in a fully-automatic fashion.

5. Related work

The work described in this paper is related to approaches developed for automated choreography realization.

In [12], the authors propose an approach to enforce synchronizability and realizability of a choreography. The method implementing the approach is able to automatically generate monitors, which act as local controllers interacting with their peers and the rest of the system in order to make the peers respect the choreography specification. Our notion of CD is "similar" to the notion of monitor used in [12], since CDs are able to interact with the choreography participants by also performing additional communication (i.e., the exchange of `sync` messages) to exogenously coordinate the participants interaction so to fulfill the choreography specification. However, the two synthesis methods are different. In [12], the monitors are generated through an iterative process, automatically refining their behavior. They are first obtained by generating the set of peers via choreography projection. Then, the system synchronizability and realizability is automatically checked by using equivalence checking. If one of these properties is violated, the method exploits the generated counterexample to augment the monitors with a new synchronization message. Our synthesis method automatically generates CDs out of the choreography specification by considering that the selected concrete participants are (language) equivalent to the choreography abstract participants. The notion of realizability that we use in this paper is the same as the one used in [12] in that both works leverage results on choreography realizability and its decidability that are given in [4,11]. However, the work in [12] concerns the generation of monitors that permit to realize the choreography out of a set of peers obtained through projection. A difference with the work presented in this paper is that our approach is reuse-based. This means that we synthesize CDs that allow to actually realize the choreography out of a set of third-party concrete services. These services are selected from an inventory by checking that their behavior is a refinement, and not necessarily the same, of the behavior of the corresponding abstract participants obtained via projection.

For what concerns synchronizability in [4,10] (and similar notions in [2,11]), and for what concerns the handling of independent sequences and independent branches, we achieve the same results as in [4,10] since we inherit the approach therein. Furthermore, we leverage the notion of Explicit Flow Model that allows to transform parallel flows into independent branches possibly executing simultaneous actions, hence accounting for both interleaving and simultaneity. Thus, we achieve synchronizability also for the handling of parallel flows.

In [19], the authors address the realizability problem based on a priori verification techniques, using refinement and proof-based formal methods. They consider asynchronous systems where peers communicate via possibly unbounded FIFO buffers. The formalized approach is different from ours in that it defines and realizes a notion of peer projection that is correct by construction, hence avoiding correct coordination from outside and, hence, avoiding the introduction in the system of additional software entities that are responsible for enforcing realizability, such as our CDs. In fact, in [19], distributed peer behaviors are first obtained via choreography projection in such a way that their synchronous composition

realizes the choreography specification. Then, the synchronous system is automatically refined into its asynchronous version where every peer is augmented by a possibly unbounded FIFO queue. The obtained asynchronous system is correct by construction, meaning that it realizes the choreography specification. Being based on a priori verification techniques and considering unbounded FIFO buffers, this method is novel and, with respect to our method and other methods discussed in this section, it is more scalable in terms of number of involved peers and exchanged messages. However, our approach focuses on realizing a choreography specification by reusing third-party peers (possibly black-box) published on a service inventory, rather than generating the correct peers from scratch. This is why we cannot avoid to deal with exogenous coordination by means of additional software entities such as the CDs.

The approach in [20] checks the conformance between the choreography specification and the composition of participant implementations. The described framework can model and analyze compositions in which the interactions can be asynchronous and the messages can be stored in unbounded queues and reordered if needed. Following this line of research, the authors provided a hierarchy of realizability notions that forms the basis for a more flexible analysis regarding classic realizability checks [20,21]. In [10], the authors identify a class of systems where choreography conformance can be efficiently checked even in the presence of asynchronous communication. This is done by checking choreography synchronizability. A set of services is synchronizable if and only if the ordering of message exchanges remain the same when asynchronous communication is replaced with synchronous communication. All these approaches are novel in that they characterize relevant properties to check choreography realizability that would represent the starting point for choreography realizability enforcement. However, they are focused on a fundamentally different problem from ours, i.e., they statically check realizability and do not account for its automatic enforcement at run time.

The ASTRO toolset supports automated composition of Web services and the monitoring of their execution [22]. It aims to compose a service starting from a business requirement and the description of the protocols defining available external services. More specifically, a planner component automatically synthesizes the code of a centralized process that achieves the business requirement by interacting with the available external services. Unlike our approach, ASTRO deals with centralized orchestration-based business processes rather than fully decentralized choreography-based ones.

The CIGAR (Concurrent and Interleaving Goal and Activity Recognition) framework aims for multigoal recognition [23]. CIGAR decomposes an observed sequence of multigoal activities into a set of action sequences, one for each goal, specifying whether a goal is active in a specific action. Although such goal decomposition somehow recalls our choreography decentralization, goal recognition represents a fundamentally different problem compared to realizability enforcement. Goal recognition concerns learning a goal-based model of an agent by observing the agent's actions while interacting with the environment. In contrast, realizability enforcement produces a decentralized coordination logic out of a choreography specification.

In [7], the authors present a unified programming framework for developing choreographies that are correct by construction in the sense that, e.g., they ensure deadlock freedom and communication safety. Developers can design both protocols and implementation from a global perspective and, then, correct endpoint implementations are automatically generated. The work in [7] considers the notion of multiparty choreography and defines choreography projection in a way that the endpoint code is correctly generated from the designed choreography. In [8], the authors discuss some of the extensions of the Jolie orchestration language developed in the context of the EU Sensoria project[11]. One of the extensions concerns the JoRBA framework that allows to develop dynamically adaptable service oriented applications. A further extension regards the Chor language that allows for the programming of distributed system from a global perspective. Chor is equipped by code generation tools that produce correct-by-construction Jolie programs, with respect to the originally designed system. The last extension is the AIOCJ choreography language that, leveraging the adaptability features of JoRBA, allows for developing adaptable choreographies. Differently from our approach, the focus of the works described in [7,8] is more on developing choregraphies from scratch rather than realizing them through the reuse of existing, often black-box, participants. However, they address choreography adaptability and flexibility [9] that allow to avoid to re-synthesize a choreography from scratch when a change occurs. Adaptability and flexibility are dimensions that are out of scope of the work presented in this paper.

6. Conclusions and future work

With the aim to rigorously assess our approach to the synthesis of service choreographies [13–15], in this paper, we have provided a formalization of the CHOReVOLUTION synthesis method to the distributed enforcement of choreography realizability. It takes as input a choreography specification in the form of a state machine and automatically generates a set of CDs. The CDs collaborate with each other to coordinate the global interaction of the selected participant services so to realize the specified choreography in a fully distributed way. The synthesized CDs are correct by construction. That is, the choreographed system resulting from the composition of the selected concrete participants and the CDs is equivalent to the choreography specification, under a suitable notion of language equivalence. We formally stated correctness of our synthesis method. The method accounts for hybrid concrete choreography participants that can perform both synchronous and asynchronous behavior. The synthesized CDs deal with asynchrony by using 1-bounded message queues. Since we deal with concrete services, we assume the existence of a service inventory where the behavioral models of the concrete

[11] www.sensoria-ist.eu.

participant services are published so to support their selection with respect to the behavior of the abstract participants in the choreography specification. Note that the assumption on the existence of behavioral models of the concrete service protocols is in practice supported by the existence of approaches to mine behavioral protocols such as Strawberry [24] and others [25,26]. Concerning the selection step, our method checks whether the behavioral model of a concrete participant is (language) equivalent to the one of an abstract participant as obtained through choreography projection.

Contribution – The main contribution of the work described in this paper, with respect to our previous work in [13–15], concerns the rigorous assessment of the synthesis method and its underlying principles, and the proof of its correctness.

Validation and experiments – To ease the comprehension of the synthesis method and its formalization, we have illustrated the method at work on a simple, yet significant, explanatory example. The low-level model that we use in this paper for choreography specification purposes can be easily built out of choreography models expressed in terms of more practical specification notations, e.g., BPMN2 Choreography Diagrams. In this direction, in our previous work [13–15], we addressed the applicability of the synthesis method, and its empirical validation, in a more practical setting. In particular, within the EU H2020 CHOReVOLUTION project (a follow-up of its predecessor EU FP7 CHOReOS project), we have applied a BPMN2-based implementation of the method to several case studies in the domains of Urban Traffic Coordination (UTC) and Smart Mobility & Tourism (SMT). The implementation of the method is part of a model-based tool chain released under the FISSi initiative[12] to support the development of choreography-based systems in CHOReVOLUTION. The implementation of our method, and of the UTC and SMT case studies, are available at: `https://tuleap.ow2.org/projects/chorevolution/`. Further details can be also found in [14], in the CHOReVOLUTION web site[13] and in the CHOReVOLUTION Platform Documentation site[14]. We applied our implementation also to a case study from the military domain concerning an instance of an Emergency Deployment System (EDS) inspired by the scenario in [27], which is representative of a large number of modern distributed software applications. We implemented a simulation of this system and used it to validate our implementation of the method. A detailed description of the case study, including a report on our findings from this work, detailed results, and related implementation, is available at: choreos.disim.univaq.it/downloads. A brief description of a further case study can be found in [28]. The application of the method to the considered case studies has shown that our approach is viable and it can be used by choreography developers.

Future improvements – As future work, in order to improve the applicability of our approach, we plan to enhance the notion of CD that we formalized in this paper in order to perform also interface and protocol adaptation beyond pure protocol coordination. Similarly, we want to investigate participant-specific solutions (e.g., discard, rollback and retry, etc.) that can be adopted to deal with independent branches and parallel flows from the point of view of the CDs actual implementation. Thus, we plan to extend our approach by parameterizing it with respect to a set of possible participant-specific solutions that can be adopted. Clearly, this would at least require to enhance the behavioral description of the concrete participants in order to include the necessary information that can enable fully-automated synthesis of the CDs actual code. Last but not least, enabling the dynamic evolution of the synthesized choreography through the synthesis of dynamically adaptable CDs represent an additional stream of future investigation.

Acknowledgements

This research work has been supported by (i) the European Union's H2020 Programme under grant agreement number 644178 (project CHOReVOLUTION – Automated Synthesis of Dynamic and Secured Choreographies for the Future Internet), (ii) the Ministry of Economy and Finance, Cipe resolution No. 135/2012 (project INCIPICT – INnovating CIty Planning through Information and Communication Technologies), and (iii) the GAUSS national research project, which has been funded by the MIUR under the PRIN 2015 program (Contract 2015KWREMX).

References

[1] European Commission, Digital Agenda for Europe – Future Internet Research and Experimentation (FIRE) Initiative, 2015.
[2] P. Poizat, G. Salaün, Checking the realizability of BPMN 2.0 choreographies, in: Proc. of SAC 2012, 2012.
[3] G. Gössler, G. Salaün, Realizability of choreographies for services interacting asynchronously, in: FACS, in: LNCS, vol. 7253, 2012, pp. 151–167.
[4] S. Basu, T. Bultan, M. Ouederni, Deciding choreography realizability, POPL, ACM, 2012.
[5] M. Güdemann, P. Poizat, G. Salaün, A. Dumont, Verchor: a framework for verifying choreographies, in: FASE, in: LNCS, vol. 7793, 2013, pp. 226–230.
[6] M. Autili, D. Ruscio, A. Di Salle, P. Inverardi, M. Tivoli, A model-based synthesis process for choreography realizability enforcement, in: FASE, in: LNCS, vol. 7793, 2013.
[7] M. Carbone, F. Montesi, Deadlock-freedom-by-design: multiparty asynchronous global programming, in: The 40th Annual ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages, POPL '13, Rome, Italy, 23–25 January 2013, 2013, pp. 263–274.
[8] I. Lanese, F. Montesi, G. Zavattaro, The evolution of Jolie – from orchestrations to adaptable choreographies, in: Software, Services, and Systems – Essays Dedicated to Martin Wirsing on the Occasion of His Retirement from the Chair of Programming and Software Engineering, 2015, pp. 506–521.
[9] M. Reichert, B. Weber, Enabling Flexibility in Process-Aware Information Systems – Challenges, Methods, Technologies, Springer, 2012.

[12] http://www.ow2.org/view/Future_Internet.
[13] www.chorevolution.eu.
[14] www.chorevolution.eu/bin/view/Documentation/WebHome.

[10] S. Basu, T. Bultan, Choreography conformance via synchronizability, in: Proceedings of the 20th International Conference on World Wide Web, WWW '11, 2011.
[11] S. Basu, T. Bultan, Automated choreography repair, in: FASE, 2016, pp. 13–30.
[12] M. Güdemann, G. Salaün, M. Ouederni, Counterexample guided synthesis of monitors for realizability enforcement, in: S. Chakraborty, M. Mukund (Eds.), Automated Technology for Verification and Analysis, in: LNCS, 2012, pp. 238–253.
[13] M. Autili, M. Tivoli, Distributed enforcement of service choreographies, in: Proceedings 13th International Workshop on Foundations of Coordination Languages and Self-Adaptive Systems, FOCLASA 2014, Rome, Italy, 6th September 2014, 2015, pp. 18–35.
[14] M. Autili, A.D. Salle, A. Perucci, M. Tivoli, On the automated synthesis of enterprise integration patterns to adapt choreography-based distributed systems, in: Proceedings 14th International Workshop on Foundations of Coordination Languages and Self-Adaptive Systems, FOCLASA 2015, Madrid, Spain, 5th September 2015, 2015, pp. 33–47.
[15] M. Autili, P. Inverardi, M. Tivoli, Automated synthesis of service choreographies, IEEE Softw. 32 (1) (2015) 50–57.
[16] S.D. Brookes, C.A.R. Hoare, A.W. Roscoe, A theory of communicating sequential processes, J. ACM 31 (3) (1984) 560–599.
[17] B. Finkbeiner, Language Containment Checking with Nondeterministic BDDs, Springer, Berlin, Heidelberg, 2001, pp. 24–38.
[18] L. Lamport, Time clocks, and the ordering of events in a distributed system, Commun. ACM 21 (1978) 558–565, https://doi.org/10.1145/359545.359563.
[19] Z. Farah, Y. Ait-Ameur, M. Ouederni, K. Tari, A correct-by-construction model for asynchronously communicating systems, Int. J. Softw. Tools Technol. Trans. 19 (4) (2017) 465–485.
[20] R. Kazhamiakin, M. Pistore, Choreography conformance analysis: asynchronous communications and information alignment, in: Web Services and Formal Methods, in: LNCS, vol. 4184, 2006, pp. 227–241.
[21] R. Kazhamiakin, M. Pistore, Analysis of realizability conditions for web service choreographies, in: Proc. of FORTE'06, in: LNCS, vol. 4229, 2006, pp. 61–76.
[22] M. Trainotti, M. Pistore, G. Calabrese, G. Zacco, G. Lucchese, F. Barbon, P. Bertoli, P. Traverso, Astro: supporting composition and execution of web services, in: ICSOC'05, in: LNCS, vol. 3826, 2005, pp. 495–501.
[23] D.H. Hu, Q. Yang, CIGAR: concurrent and interleaving goal and activity recognition, in: Proc. of AAAI'08, 2008, pp. 1363–1368.
[24] A. Bertolino, P. Inverardi, P. Pelliccione, M. Tivoli, Automatic synthesis of behavior protocols for composable web-services, in: Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering, Amsterdam, The Netherlands, 24–28 August 2009, 2009, pp. 141–150.
[25] M. Autili, D. Di Ruscio, P. Inverardi, P. Pelliccione, M. Tivoli, Modelland: where do models come from?, in: Models@runtime, in: LNCS, vol. 8378, Springer, 2014.
[26] S. Uchitel, G. Brunet, M. Chechik, Synthesis of partial behavior models from properties and scenarios, IEEE Trans. Softw. Eng. 35 (3) (2009) 384–406.
[27] S. Malek, N. Beckman, M. Mikic-Rakic, N. Medvidovic, A framework for ensuring and improving dependability in highly distributed systems, in: Architecting Dependable Systems III, 2004, pp. 173–193.
[28] M. Autili, K.L. Bellman, A. Diaconescu, L. Esterle, M. Tivoli, A. Zisman, Transition Strategies for Increasing Self-awareness in Existing Types of Computing Systems, 2017.