Improving the automatic derivation of choreography-conforming web services systems

Pablo Rabanal,Ismael Rodríguez,Jose A. Mateo and Gregorio Díaz

December 10, 2013

Abstract

We present DIEGO 2.0, a new improved version of our tool DIEGO. Given a WS-CDL choreography, it automatically extracts a set of WS-BPEL web services such that, if these services interact with each other, they necessarily produce the behavior defined in the choreography even if the natural projection does not work. This is achieved by introducing some control messages which make services coordinate as expected. The main improvement with respect to the previous version is the number of these messages has been dramatically reduced. We formally define the new derivation, prove its correctness, and empirically compare the efficiency of new and former derivation algorithms. We also introduce other improvements of the new tool version, such as the testing engine.

1 Introduction

The definition of a web service-oriented system involves two complementary views: *Orchestration* and *choreography*. The orchestration concerns the *internal* behavior of a web service in terms of invocations to other services. It is supported, e.g., by WS-BPEL (*Web Services Business Process Execution Language*), which is considered the de-facto standard language for describing the workflow of a web service. On the other hand, the choreography concerns the *observable* interaction among web services. It can be defined, e.g., by using WS-CDL (*Web Services Choreography Description Language*). Thus, the collaborative behavior, described by the choreography, should be the result of the interaction of the individual behaviors of each involved party, which are defined via the orchestration.

In this paper, we present DIEGO 2.0, a new version of our tool for *DerIving chorEoGraphy-cOnforming web service systems*. Given a choreography defined in (a subset of) WS-CDL, DIEGO automatically extracts a set of services defined in WS-BPEL such that the interaction of these services necessarily leads to the behavior defined by the choreography. Although DIEGO transforms WS-CDL into WS-BPEL, it internally works with an modification of *finite state machines* (FSMs). First, DIEGO transforms the WS-CDL choreography into a variant of FSM where involved services are explicitly identified. Next it extracts, from this FSM-based model, services defined by means of a different kind of FSMs variant where input buffers are used to support

asynchronous communications. Finally, these FSM extensions are transformed into WS-BPEL. In our model, the WS-CDL constructions represented are interaction, sequence, and while, leaving as future work the inclusion of other constructions such as the parallel operator and the workunits.[1]

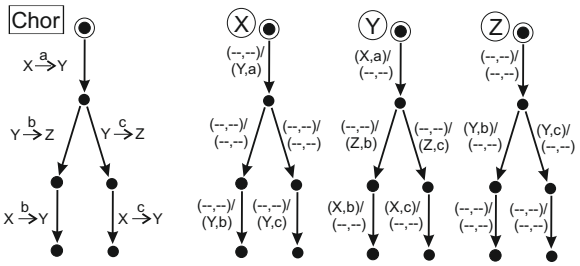


Figure 1: Example of a natural projection.

The main problem arisen in the derivation of services from a choreography is the fact that the *natural projection* does not necessarily produce a set of services conforming to the choreography. In the natural projection (see Figure 1), the structure of states and transitions of the choreography ($Chor$) is directly copied into derived services (X, Y, Z). Therefore, if the choreography has a transition from state s_1 to s_2 where service X sends message a to service Y, then the derived service X has a transition from s_1 to s_2 where it sends a to Y, the derived service Y has a transition from s_1 to s_2 where it receives a from X, and all the rest of derived services (not involved in this choreography step) have a null transition from s_1 to s_2, i.e. they change their state doing nothing (Z in Figure 1). It is known that this straightforward derivation scheme produces sets of derived services that could *not* behave as defined by the choreography. On the one hand, derived services could tackle non-determinism in an inconsistent way when running together. In a choreography state where several transitions (choices) are available, each service could follow a *different* path, thus leading to inconsistent subsequent states in each service. On the other hand, services which are not involved in the current step could silently evolve to some subsequent state and send messages as required by subsequent transitions *before* the services involved in the current step actually do, thus incorrectly overtaking them. Moreover, since the communication medium may delay messages any arbitrarily long time, it may happen that a message m' is sent after message m, but m' arrives to its destination service before m, yielding a race condition.

In previous works [5, 6, 13], theoretical FSM-based models for defining orchestrations and choreographies are presented. Some conformance relations to decide whether a set of services necessarily produces the behavior defined by a choreography are presented, and derivation algorithms to automatically extract the former from the latter are given. These algorithms tackle the problems mentioned before by adding some control messages which make services coordinate as expected by the choreography. The

[1]Thus, in our language model, the concurrency is the result of the parallel interaction between *different* services, but each service does not contain parallel processes internally. Still, this model captures the conceptual difficulty of the deriving services that correctly interact *with each other*, which is our goal.

correctness of these algorithms with respect to these conformance relations is formally proved.

Unfortunately, the original version of DIEGO [12] has some important limitations. The most important one is the big number of additional messages exchanged between services in order to make them correctly coordinate. In each choreography step, coordinating messages are sent to *all* services in order to make them mimic the path of the choreography being executed. However, this scheme can be modified in order to avoid sending messages to services which are not involved in the current step. In particular, non-involved services can be informed *later* about what choreography state they should be in, only when they are involved *again* in the step under execution. This change complicates the definition of the derivation algorithms, but strongly improves the efficiency of derived services.

Besides, the new version allows users to systematically analyze any alternative derivation algorithm. After introducing the new algorithm, the tool automatically tests the system of derived services by randomly applying the language operational semantics (i.e. it randomly chooses the execution path among all available execution paths) and compares the observations with those permitted by the choreography under different conformance requirements: *Sending* (that is, with respect to moments when messages are sent), *processing* (w.r.t. moments when messages are processed), and *synchronization* (both). Moreover, in order to improve the observability of derived systems, they can be executed either step by step or immediately upon termination (or upon some point), and simulations are exported to text files. The main advantage of using DIEGO is that web service designers can rapidly obtain executable prototypes from a choreography definition, which may help them to debug the system in early stages of the development process or even serve as executable skeletons for constructing the implementations from them. DIEGO 2.0 is available at `http://www.dsi.uclm.es/retics/diego/`.

The rest of the paper is organized as follows. Below, we review some related works. In Section 3 we formally define our *centralized* and *decentralized* derivation algorithms. In Section 4 we report experiments where we compare the performance of services derived by our enhanced algorithms with the performance of previous derivation algorithms finishing the paper with some conclusions and future work. Proofs are presented in the appendixes.

2 Related Work

In this section we briefly compare our proposal with some works related to ours. We classify related proposals into two main groups: Works describing transformations which have been implemented but not formally defined and verified, and proposals based upon rigorous transformations, generally using some kind of formal model.

Regarding implemented transformations, one of the most widely spread proposals in the community is the work proposed by Mendling and Hafner [10]. In this work, authors propose a transformation between the different elements of WS-CDL into WS-BPEL. Furthermore, a prototype is implemented as a proof of concept of the proposal. The main disadvantage of this work is that authors do not provide any mechanism to

verify either the correction of the generated stubs or when the generated services behave as intended in the choreography. This is a common lack in tools and prototypes providing similar transformations, typically due to the fact that they are not formally defined. An example of this is found in the proposal by Khadka et al [7], where authors describe the transformation from WS-CDL into WS-BPEL by using the ATLAS transformation Language.

Next we consider other proposals based upon formally descriptions. Most of approaches in the literature study the conditions that are necessary to make the natural projection work, i.e. to make naturally projected services necessarily work as defined in the choreography. On the contrary, we consider that the previous problems can be fixed if appropriate control messages are added to make services correctly coordinate. Thus, we consider that *all* choreographies are realizable indeed. In [11], Zongyan et al. identify and face the problems appearing when deriving an implementable projection from a choreography. Authors define the concept of restricted natural choreography, which must fulfill two structural conditions, and show that this kind of choreography is easily implementable. A new concept, the *dominant role* of a choice, is proposed for dealing with projection in non-restricted choreographies. At each non-deterministic choice, this dominant role is the one that makes the decision.

Salaün and Bultan [14] formalize choreographies by means of asynchronous communication with process algebra. However, no solution for non-deterministic choices is provided and no correctness proof is presented. In contrast, authors enhance the proposal by introducing a tool offering the possibility to use bounded buffers and reason about them. Van der Aalst et al. [16] present an approach for formalizing compliance and refinement notions, which are applied to service systems specified using Open Workflow Nets (a type of Petri Nets) where the communication is asynchronous. A similar approach is followed by Caires and Vieira [3]. They define a formal framework called conversation types and present techniques to ensure progress of systems involving several interleaved conversations/sessions. Bravetti and Zavattaro [2] compare systems of orchestrations and choreographies by means of a *testing* relation. Systems are represented by using a process algebraic notation, and operational semantics for this language are defined as LTS.

In [9], Lucchi and Mazzara provide a formalization of conformance with π-calculus. By means of automata, Baldoni et al. [1] define a conformance notion that checks whether the interoperability is guaranteed. Moreover, Decker et al. [4] show how the Business Process Modeling Notation (BPMN) and the Business Process Execution Language (BPEL) can be used during choreography design. In another work [15], Van der Aalst et al. also focus on conformance by comparing the observed behavior with some predefined model.

Lanese et al. in [8] developed a very detailed and broad study to compare these kind of systems. Their objective is bridging the gap between the WS-CDL and BPEL languages by formally defining them and then finding out the *features* systems should have to be equivalent if the natural projection is used. This work is based on the idea of *well formed conditions*, which depend on the properties one wants to preserve in each case. Table 1 presents a comparison of the relations proposed by this work and FSM-based conformance relations given in [13].

Let us note that most of the previous works are only theoretical. On the contrary, in

	synchr.	asynchr.	send. vs proc.	full vs partial	send + proc	disjoint	immediate vs delayed
Our work	X	✓	✓	✓	✓	X	✓
Lanese et al.	✓	✓	✓	X	✓	✓	X

Table 1: Comparison between Lanese et al. work and our work.

this paper we apply previous theoretical FSM-based models and develop new (formally proved) derivation algorithms to improve a *tool* that automatically transforms WS-CDL choreographies into WS-BPEL orchestrations.

There are some tools that formally define either WS-CDL or WS-BPEL such as WS-Engineer[2], XFD-DEVS[3], WSMod, WSAT, WS-VERIFY, PI4SOA. They focus on the analysis of the models obtained from the formalization process, but they are not concerned about the kind of transformations given in our proposal.

3 Deriving Conforming Services

In this section we present our optimized centralized and decentralized derivation algorithms. The algorithms are defined by using some auxiliary notions from previous works [5, 6, 13]. Due to the lack of space, we focus here on formally presenting the new algorithms and informally sketching the rest of necessary notions.

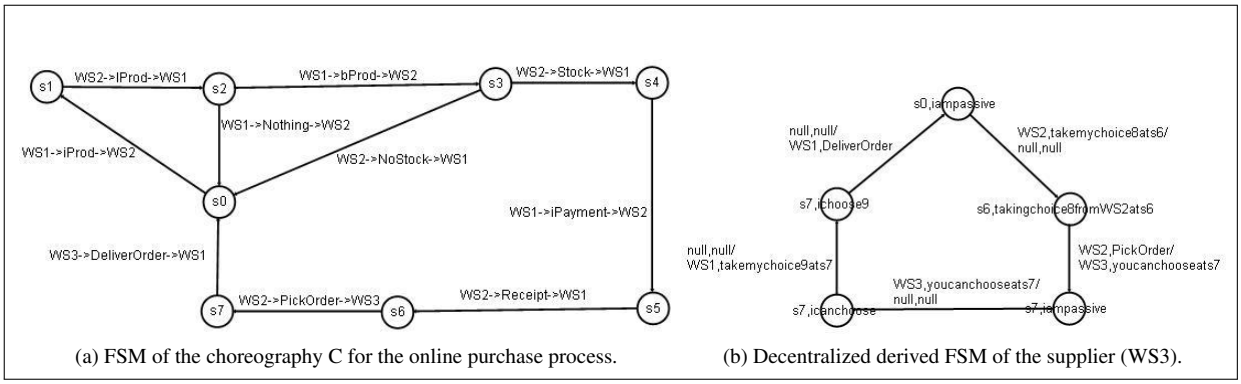
In order to illustrate the application of the proposed notions to a system, we present a small case study that will guide the presentation of some concepts and will serve as the basis for the definition of examples. This case study is a typical purchase process that uses Internet as a business context for a transaction. There are three actors in this example: a customer, a seller, and a supplier. The purchase works as follows: *"A customer wants to buy a product by using Internet. There are several sellers that offer different products in web-pages servers. The customer contacts a seller in order to buy the desired product. The seller checks the stock and contacts a supplier. Finally, the supplier delivers the product to the customer."* The behavior of each participant is defined as follows:

- Customer: It contacts the seller to buy a product. After consulting the product list, it can either order a product or do nothing. If the customer decides to buy a product, then it must send the seller the information about the product and the payment method. After the payment is done, it waits to receive the product from a supplier.
- Seller: It receives the customer order and the payment method. The seller checks if there is enough stock to deliver the order and sends an acceptance notification to the customer. If there is stock to deliver the order, then it contacts a supplier to deliver the product.
- Supplier: It gathers the order and the customer information in order to deliver the product to the customer.

[2]Available at: `http://www.doc.ic.ac.uk/ltsa/eclipse/wsengineer/`
[3]Available at: `http://duniptechnologies.com/research/xfddevs/`

Let us introduce the main notions of the formal framework:

FSM-based model for the choreography: The choreography model focuses on representing the interaction of services as a whole. It is defined as a tuple (S, M, ID, s_{in}, T) where S denotes the set of states, M is the set of messages, ID is the set of service identifiers, $s_{in} \in S$ is the initial state, and T is the set of transitions. A transition $t \in T$ is a tuple (s, m, snd, adr, s') where $s, s' \in S$ are the initial and final states, respectively, $m \in M$ is the message, and $snd, adr \in ID$ are the sender and the addressee of the message, respectively. A transition (s, m, snd, adr, s') is also denoted by $s \xrightarrow{m/(snd \to adr)} s'$. Each choreography transition denotes a message action where some service sends a message to another one. The actual configuration of a choreography consists just in the current state of its FSM model. In Figure 2a, the choreography C of our Internet purchase process is shown, and it is defined as $(S_{2a}, M_{2a}, ID_{2a}, s_0, T_{2a})$ where $S_{2a} = \{s0, s1, \ldots\}$, $M_{2a} = \{iProd, lProd, \ldots\}$, $ID_{2a} = \{WS1, WS2, WS3\}$, $s0$ is the initial state and $T_{2a} = \{(s0, iProd, WS1, WS2, s1), (s1, lProd, WS2, WS1, $ In the example, $WS1$ is the customer, $WS2$ represents the seller, and $WS3$ the supplier.



(a) FSM of the choreography C for the online purchase process. (b) Decentralized derived FSM of the supplier (WS3).

Figure 2: FSMs automatically generated with DIEGO.

FSM-based model for the orchestration: The behavior of a web service in terms of its interaction with other web services is represented by a finite state machine (id, S, I, O, s_{in}, T) where $id \in ID$ is the identifier of the service and ID is the set of service identifiers, S is the set of states, I is the set of input messages, O is the set of output messages, $s_{in} \in S$ is the initial state, and T is the set of transitions. Each transition $t \in T$ is a tuple (s, i, snd, o, adr, s') where $s, s' \in S$ are the initial and final states respectively, $i \in I$ is the message that must be received to trigger the transition, $snd \in ID$ is the required sender of i, $o \in O$ is the output message which is sent if the transition is taken, and $adr \in ID$ is the addressee of o. A transition (s, i, snd, o, adr, s') is also denoted by $s \xrightarrow{(snd,i)/(adr,o)} s'$. Each service is endowed with an input buffer to store messages received but not yet processed, so this transition is triggered if there is a message i from snd in the service input buffer. The actual configuration of an orchestration consists of two elements: its current state and the contents of its input buffer. Transitions can also be triggered without consuming any input (i.e. proactively) or without producing any

output. This is denoted in our formalism by using a $null$ $(--, --)$ input/output symbol, respectively. In Figure 2b, a web service definition for the supplier is shown, and it is defined as $(WS3, S_{2b}, I_{2b}, O_{2b}, s0iampassive, T_{2b})$ where $WS3$ is the name of the orchestration, $S_{2b} = \{s0iampassive, s7ichoose9, \ldots\}$, $I_{2b} = \{takemychoice8ats6, PickOrder, \ldots\}$, $O_{2b} = \{youcanchooseats7, DeliverOrder, \ldots\}$, $s0iampassive$ is the initial state, and $T_{2b} = \{(s0iampassive, takemychoice8ats6, WS2, null, null, s6takingchoice8fromWS2ats$ $(s6takingchoice8fromWS2ats6, PickOrder, WS2, youcanchooseats7, WS3, s7iampassive), \ldots\}$. In fact, this is the supplier service definition automatically extracted from the choreography by our decentralized service derivation algorithm, which will be presented later.

In both models, *traces* (i.e. sequences of events that can be produced by systems) are represented in a similar way, i.e., each event is denoted by a message, a sender, and an addressee (snd, m, adr). However, in the orchestration model, events may refer to either the moments when messages are *sent* or the moments when messages are *processed* by their corresponding receivers (both moments differ because we consider an asynchronous environment). The operational semantics, which defines how orchestrations and choreographies evolve step-by-step, is given in [5, 13]. In particular, the semantics given in [13], assumed in this paper, allows each message to be delayed in the communication medium for any arbitrarily long time. Based on the traces of choreographies and orchestrations, formal relations (*conformance relations*) are defined to determine whether a system of orchestrations necessarily produces the behavior required by a choreography. Several relations are given. On the one hand, the conformance of the system of services may be established in terms of *sending* actions ($\texttt{conf}_s$), *processing* actions ($\texttt{conf}_p$) or both ($\texttt{conf}$). On the other hand, systems of services may be required to be able to perform *all* non-deterministic paths defined in the choreography (the superscript f is added to the relation symbol), or only at least *one* of them (without superscript). As a result, the conformance relations $\texttt{conf}_s$, $\texttt{conf}_p$, $\texttt{conf}$, $\texttt{conf}_s^f$, $\texttt{conf}_p^f$, $\texttt{conf}^f$, as well as their corresponding *prime* counterparts (i.e. $\texttt{conf}'_s, \ldots$, where control messages are ignored), are defined formally in [5, 13].

As commented previously, the problems of the natural projection (inconsistent non-deterministic choices and race conditions), cannot be solved unless we introduce some additional *control messages* to be in charge of coordinating services as required by the choreography. In this way, systems conforming to the choreography (up to *original* choreography messages, i.e. ignoring additional control messages) can be derived. Two optimized derivation methods, *centralized* and *decentralized*, are now presented. First, an auxiliary notion is introduced.

Definition 3.1 Let $\mathcal{C} = (S, M, ID, s^1, T)$ be a choreography. Let $s \xrightarrow{m_1/(a_1 \to b_1), \ldots, m_n/(a_n \to b_n)} s'$ be a shorthand for $\exists\ s_1, \ldots, s_{n-1} : s \xrightarrow{m_1/(a_1 \to b_1)} s_1 \in T, s_1 \xrightarrow{m_2/(a_2 \to b_2)} s_2 \in T, \ldots, s_{n-1} \xrightarrow{m_n/(a_n \to b_n)} s' \in T$. For all state $s \in S$ and $id \in ID$, $\texttt{upToSnd}(\mathcal{C}, s, id)$ is defined as

$$\left\{ s' \;\middle|\; ((s \xrightarrow{m_1/(a_1 \to b_1), \ldots, m_n/(a_n \to b_n)} s' \wedge a_1, b_1, \ldots, a_n, b_n \neq id) \vee s = s') \wedge s' \xrightarrow{m_{n+1}/(id \to b_{n+1})} s'' \in T \right\}$$

representing the set of choreography states s' we can reach from state s in such a way that the service id acts as *sender* in s' but does not participate in the intermediate

transitions from s to s' either as sender or as receiver. And $\texttt{upToRcv}(\mathcal{C}, s, id)$ is defined as

$$\left\{ s' \;\middle|\; ((s \xrightarrow{m_1/(a_1 \to b_1),\ldots,m_n/(a_n \to b_n)} s' \;\wedge\; a_1, b_1, \ldots, a_n, b_n \neq id) \;\vee\; s = s') \;\wedge\; s' \xrightarrow{m_{n+1}/(a_{n+1} \to id)} s'' \in T \right\}$$

representing the set of choreography states s' we can reach from state s in such a way that the service id acts as *receiver* in s' but does not participate in the intermediate transitions from s to s' either as sender or as receiver. □

Next we define our new derivation algorithms to extract service orchestrations from choreographies. First we define the *centralized* derivation method. A centralized solution may look counter-intuitive to model web services, but there are specific systems where a centralized solution is needed since some decisions depend on a unique service. For example, multiplayer online games are played via a centralized game server, and centralized peer-to-peer systems are based on a central server which is used for indexing functions and bootstrap the entire system. In the centralized derivation, all services are naturally projected from the choreography, but some projected transitions are deleted whereas others are added. Any sequence of inactive (null) transitions in the projection which ends with an effective transition (i.e. a sending or a reception) is replaced by a single transition doing only the effective action, allowing the service to skip all pointless transitions during its idle period. A new service, called *orchestrator*, is added to decide the path in each non-deterministic choice of the choreography. It propagates its decisions to the services involved in the selected transition and makes services not overtake each other. This is achieved by making the orchestrator and the rest of services exchange some control messages. In order to avoid a wrong processing ordering of messages in different choreography steps, the orchestrator blocks those services trying to go further than the orchestrator itself. It is worth noting that, since null projected transitions are deleted in the rest of services, the orchestrator must be able to coordinate services that became idle at *different* previous choreography states. This complicates the algorithm definition but strongly improves the system efficiency, as we will illustrate in Section 4.

Definition 3.2 Let $\mathcal{C} = (S, M, ID, s^1, T)$ be a choreography machine with $ID = \{id_1, \ldots, id_n\}$ and $S = \{s^1, \ldots, s^l\}$. For all $s \in S$ and $id \in ID$, let $T_{s,id} = \{s \xrightarrow{m/(id \to adr)} s' \mid \exists\, adr, m, s' : s \xrightarrow{m/(id \to adr)} s' \in T\}$. For all $1 \leq j \leq |T_{s,id}|$, let $t_{s,id,j}$ denote the j-th transition of $T_{s,id}$ according to some arbitrary ordering criterium.

For all $1 \leq i \leq n$, the *controlled service* for $\mathcal{C}$ and id_i, denoted $\texttt{controlled}(\mathcal{C}, id_i)$, is a service $M_i = (id_i, S'_i, I'_i, O'_i, s^1_{iampassive}, T_i)$, where S'_i, I'_i, O'_i consist of all states, inputs, and outputs appearing in transitions described next and, for all $s^q \in S$, the following transitions are in T_i:

- For all $s^r \in S$ such that $s^q \in \texttt{upToRcv}(\mathcal{C}, s^r, id_i)$ *(i.e. there is a path from s^r to s^q where id_i does nothing, but id_i may receive a message at s^q)* and for all $t = s^q \xrightarrow{m/(snd \to id_i)} s'^q \in T$ *(i.e. for all transitions where id_i is the receiver at s^q)* we have the following transitions, where we assume $t = t_{s^q,snd,j}$:

(i) $s^r_{iampassive} \xrightarrow{t_1} s^q_{takingchoice_j of_{snd} at_{s^q}}$ where $t_1 = (orc, takechoice_j of_{snd} at_{s^q})/(null, null)$ *(the orchestrator tells id_i that id_i will be the receiver at this state, as well as the choice it must take).*

(ii) $s^q_{takingchoice_j of_{snd} at_{s^q}} \xrightarrow{t_2} s'^q_{iampassive}$ where $t_2 = (snd, m)/(orc, received)$ *(id_i receives the message and tells the orchestrator that it can initiate next transition).*

- If $|T_{s^q,id_i}| \geq 1$ *(that is, id_i is a service that sends a message at, at least, one transition leaving the choreography state s^q)* then, for all $1 \leq j \leq |T_{s^q,id_i}|$, we have the following transitions, where we assume $t_{s^q,id_i,j} = s^q \xrightarrow{m/(id_i \to adr)} s'^q$:

(i) For all $s^r \in S$ such that $s^q \in \texttt{upToSnd}(\mathcal{C}, s^r, id_i)$ we have the transition $s^r_{iampassive} \xrightarrow{t_3} s'^q_{iampassive}$ where $t_3 = (orc, takeyourchoice_j at_{s^q})/(adr, m)$ *(the orchestrator tells the service that its j-th sending transition will be taken and the service sends the message of that transition).*

The *orchestrator* of $\mathcal{C}$, denoted by $\texttt{orchestrator}(\mathcal{C})$, is a service $O = (orc, S', I', O', s^1, T')$, where S', I', O' consist of all states, inputs, and outputs appearing in transitions described next and, for all $s^q \in S$, $snd \in ID$, and $1 \leq j \leq |T_{s^q,snd}|$ we have the following transitions in T', where we assume $t_{s^q,snd,j} = s^q \xrightarrow{m/(snd \to rcv)} s'^q$:

- $s^q \xrightarrow{t_4} s^q_{startingchoice_j of_{snd}}$ where $t_4 = (null, null)/\ (snd, takeyourchoice_j at_{s^q})$ *(the orchestrator tells the sender that one of its transitions will be taken, as well as which one).*

- $s^q_{startingchoice_j of_{snd}} \xrightarrow{t_5} s^q_{takingchoice_j of_{snd}}$ where $t_5 = (null, null)/(rcv, takechoice_j of_{snd} at_{s^q})$ *(the orchestrator tells the receiver that one of the transitions of snd will be taken, as well as which one).*

- $s^q_{takingchoice_j of_{snd}} \xrightarrow{t_6} s'^q$ where $t_6 = (rcv, received)/(null,\ null)$ *(the orchestrator receives the acknowledgement from the receiver and the choreography transition is finished).*

□

Theorem 3.3 Let $\mathcal{C} = (S, M, ID, s_{in}, T)$ be a choreography with $ID = \{id_1, \ldots, id_n\}$. Let $\mathcal{S} = (\texttt{controlled}(\mathcal{C}, id_1),$ $\ldots, \texttt{controlled}(\mathcal{C}, id_n), \texttt{orchestrator}(\mathcal{C}))$. For all conformance relationships $\texttt{conf}_x \in \{\texttt{conf}'_s, \texttt{conf}'_p, \texttt{conf}', \texttt{conf}^{f\prime}_s, \texttt{conf}^{f\prime}_p, \texttt{conf}^{f\prime}\}$ we have $\mathcal{S}\ \texttt{conf}_x\ \mathcal{C}$.
□

Let us introduce the *decentralized* derivation algorithm. In this algorithm, the coordination responsibility is distributed among the participants. This approach is more realistic in cases where we cannot assume the existence of an almost omniscient orchestrator owning the relevant information to decide (as we need to assume in the centralized derivation). Again, services are derived by a natural projection, although

sequences of idle transitions ending with an active transition are replaced by the latter transition, and some control messages between the services themselves (not with any orchestrator) are added. For each choreography state having several outgoing transitions, we assume that services being able to send a message are ordered according to any arbitrary criterion. The first service chooses between (a) taking one of the transitions it may trigger at the current state, or (b) refusing to do so and delegating the transition election to the next potentially-deciding service. In case (b), the next service also chooses either (a) or (b), and so on until the last deciding service, which, if reached, must choose (a). When a service in the sequence chooses (a), it tells its choice to the receiver of its message and next sends the message. When the receiver gets it, it sends a message to the first deciding service of the new choreography state, meaning that the decision-making process of that new state may begin. Figure 2b shows the derived orchestration of the supplier for the online purchase process. Notice that this machine is generated automatically by DIEGO and is depicted by using the system viewer implemented in the tool. Due to the lack of space, we do not present the rest of derived services of the case study, but they can be easily extracted by using the tool, as this choreography example is included in the files included in the downloadable version of the application (..*DIEGO2.0**Other_Choreographies**chorFig6.xml*).

Definition 3.4 Let $\mathcal{C} = (S, M, ID, s^1, T)$ be a choreography machine with $ID = \{id_1, \ldots, id_n\}$ and $S = \{s^1, \ldots, s^l\}$. For all $s \in S$ and $id \in ID$, let $T_{s,id} = \{s \xrightarrow{m/(id \to adr)} s' \mid \exists\, adr, m, s' : s \xrightarrow{m/(id \to adr)} s' \in T\}$. For all $1 \leq j \leq |T_{s,id}|$, let $t_{s,id,j}$ denote the j-th transition of $T_{s,id}$ according to some arbitrary ordering criterium.

For all $s \in S$, let $[\alpha^s_1, \ldots, \alpha^s_{h_s}]$ denote any arbitrarily-ordered sequence of all identifiers in $id \in ID$ such that $|T_{s,id}| \geq 1$.

For all $1 \leq i \leq n$, the *decentralized service* for $\mathcal{C}$ and id_i, denoted $\texttt{decentral}(\mathcal{C}, id_i)$, is a service $M_i = (id_i, S'_i, I'_i, O'_i, s_{in}, T_i)$, where S'_i, I'_i, O'_i consist of all states, inputs, and outputs appearing in transitions described next and, for all $s^q \in S$, the following transitions are in T_i:

- For all $s^r \in S$ such that $s^q \in \texttt{upToRcv}(\mathcal{C}, s^r, id_i)$ *(i.e. there is a path from s^r to s^q where id_i does nothing, but id_i may receive a message at s^q)* and for all $t = s^q \xrightarrow{m/(snd \to id_i)} s'^q \in T$ *(i.e. for all transitions where id_i is the receiver at s^q)* we have the following transitions, where we assume $t = t_{s^q,snd,j}$:

  (i) $s^r_{iampassive} \xrightarrow{t_7} s^q_{takingchoice_j of_{snd}}$ where $t_7 = (snd, takemychoice_j at_{s^q})/(null, null)$ *(the service choosing at this state tells id_i that id_i will be the receiver at this state).*

  (ii) If there exists at least one transition leaving s'^q in T then we also have the transition $s^q_{takingchoice_j of_{snd}} \xrightarrow{t_8} s'^q_{iampassive}$ where $t_8 = (snd, m)/(\alpha^{s'^q}_1, youcan chooseat_{s'^q})$ *(id_i receives the message and tells the first sender of the new state that it can initiate the decision-making of the next state).*
  Else, $s^q_{takingchoice_j of_{snd}} \xrightarrow{(snd,m)/(null,null)} s'^q_{iampassive}$.

- If we have $id_i = \alpha_k^{s^q}$ such that $1 \leq k \leq h_{s^q}$ *(that is, id_i is a service that sends a message at, at least, one transition leaving the choreography state s^q)* then we have the following additional transitions, where we assume $(id_i)^- = \alpha_{k-1}^{s^q}$ and $(id_i)^+ = \alpha_{k+1}^{s^q}$:

  (a) Take the appropriate choice:

  (a.1) *If id_i is the first service deciding at s^q, i.e. $id_i = \alpha_1^{s^q}$:* Let $J = \{b \mid \exists\, s', m, a : s' \xrightarrow{m/(a \to b)} s^q \in T\}$. For all $s^r \in S$ such that $s^q \in \texttt{upToSnd}(\mathcal{C}, s^r, id_i)$ and for all $b \in J$ we have $s^r_{iampassive} \xrightarrow{t_9} s^q_{icanchoose}$ where $t_9 = (b, youcanchooseat_{s^q})/(null, null)$ *(the receiver of the previous choreography transition tells id_i that it can start the decision-making of choreography state s^q).*

  (a.2) *Else:* For all $s^r \in S$ such that $s^q \in \texttt{upToSnd}(\mathcal{C}, s^r, id_i)$ we have the transition $s^r_{iampassive} \xrightarrow{t_{10}} s^q_{icanchoose}$ where $t_{10} = ((id_i)^-, youcanchooseat_{s^q})/(null, null)$ *(the previous service which can decide at the current state tells id_i and it will not, so id_i can).*

  (b) *If id_i is not the last service deciding at s^q, i.e. $id_i \neq \alpha_{h_{s^q}}^{s^q}$:* We have the transition $s^q_{icanchoose} \xrightarrow{t_{11}} s^q_{iampassive}$ where $t_{11} = (null, null)/((id_i)^+, youcanchooseat_{s^q})$ *(id_i refuses to choose and tells the next service that it can choose)*

  (c) For all $1 \leq j \leq |T_{s^q, id_i}|$ we have the following transitions, where we assume $t_{s^q, id_i, j} = s^q \xrightarrow{m/(id_i \to adr)} s'^q$.

  (c.1) $s^q_{icanchoose} \xrightarrow{t_{12}} s^q_{ichoose_j}$ where $t_{12} = (null, null)/\ (adr, takemychoice_j at_{s^q})$ *(id_i chooses its j-th transition and informs the receiver of that transition about it).*

  (c.2) $s^q_{ichoose_j} \xrightarrow{(null,null)/(adr,m)} s'^q_{iampassive}$ *(id_i sends the message required by the choreography and moves to a passive state).*

  Besides, if $id_i = \alpha_1^{s^1}$ then $s_{in} = s^1_{icanchoose}$ else $s_{in} = s^1_{iampassive}$. □

Theorem 3.5 Let $\mathcal{C} = (S, M, ID, s_{in}, T)$ be a choreography with $ID = \{id_1, \ldots, id_n\}$. Let $\mathcal{S} = (\texttt{decentral}(\mathcal{C}, id_1), \ldots, \texttt{decentral}(\mathcal{C}, id_n))$. For all $\texttt{conf}_x \in \{\texttt{conf}'_s, \texttt{conf}'_p, \texttt{conf}', \texttt{conf}^{f\prime}_s, \texttt{conf}^{f\prime}_p, \texttt{conf}^{f\prime}\}$ we have $\mathcal{S}\ \texttt{conf}_x\ \mathcal{C}$. □

4 Experiments

In this section we compare the performance of our centralized and decentralized derivations, as well as them with derivations implemented in the former DIEGO version (defined in [13]), in terms of the number of messages exchanged among the services. We compare them by using several choreographies where the number of involved web services differs or the choreography *depth* (i.e. the depth of the tree denoting the choreography form) varies. The tool DIEGO 2.0, as well as all choreographies mentioned in experiments described here and other related material, are available at `http://www.dsi.uclm.es/retics/diego/`.

Experiments performed with the automatic testing functionality of DIEGO 2.0 show that new algorithms suppress a high amount of unnecessary control messages between the services, yielding a very efficient derivation. Decreasing the message traffic is very important in order to reduce the network congestion. We empirically observe that, in former algorithms, most of *avoidable* control messages in the derivation algorithms are sent to services that are not involved in the current choreography step indeed. On the contrary, in the optimized algorithms, for each service, any sequence of consecutive *null* transitions, departing at state s, and followed by a non-null transition reaching state s', is replaced by a *single* transition from s to s'. These transitions make the service immediately advance a *different* number of choreography steps, depending on its current state. Thus, services must be informed of the transition to be taken at each time, and also of what choreography state this transition belongs to.

Let us introduce our choreography examples. In choreography $cExp1\_2ws$, only two services participate in a single branching point. In choreography $cExp1\_3ws$, $cExp1\_2ws$ is modified by adding a new participant service, yielding a choreography where three web services participate in a single choice point. By changing the number of participant services in $cExp1\_2ws$ we create the rest of the choreographies used in the experiments, where $4, 5, \ldots, 10$ web services are involved (these choreographies are called $cExp1\_4ws, cExp1\_5ws, \ldots, cExp1\_10ws$). We compare the number of messages exchanged among the services by using our tool DIEGO 2.0, in particular, the web services simulator included in it. The results are shown in Table 2. The first column of Table 2, called *Version*, shows the algorithm version, 1.0 (former version) or 2.0 (optimized version); column *Choreography* shows the name of the choreography used in the experiment; column *Msgs Centr.* shows the number of messages exchanged between the services when running the simulation using the centralized derivation; finally, column *Msgs Decentr.* shows the number of messages exchanged between the services when running the simulation using the decentralized derivation.Regarding to the column *Msgs Decentr.*, sometimes an interval appears. This means that the derivation version under consideration needed different numbers of messages in different executions. Notice that the choreography non-determinism is simulated by making *random* choices in the simulator, so the number of messages upon termination may vary depending on the path chosen. Using the results obtained, we have characterized (in the last two lines of Table 2) the number of messages exchanged with respect to the number of web services involved in a choreography branch (p) or the choreography depth (q).

The results illustrate that the number of messages exchanged in the *former* versions of the centralized and decentralized derivation algorithms grows linearly with the number of services involved in the choice point of the choreography, whereas the number of messages exchanged in the *optimized* versions remains constant in the centralized version and in the lowest bounds of the decentralized version. So, if the number of web services increases, the number of messages exchanged is not affected in the centralized case. Furthermore, if we include more branches leaving the same choice state and the number of web services being able to *send* increases, then the number of messages required for communication increases linearly in the upper bounds in the *decentralized* version since they have to agree on which one will act. However, this does not affect the centralized version because the orchestrator makes the decision in a single step,

Version	Choreography	Msgs Centr.	Msgs Decentr.	Choreography	Msgs Centr.	Msgs Decentr.
1.0	cExp1_2ws	11	8-9	cExp2_d=3	18	18
2.0	cExp1_2ws	6	4-6	cExp2_d=3	12	9
1.0	cExp1_3ws	16	11-13	cExp2_d=4	27	27
2.0	cExp1_3ws	6	4-8	cExp2_d=4	18	14
1.0	cExp1_4ws	21	14-17	cExp2_d=5	36	36
2.0	cExp1_4ws	6	4-10	cExp2_d=5	24	19
1.0	cExp1_5ws	26	17-21	cExp2_d=6	45	45
2.0	cExp1_5ws	6	4-12	cExp2_d=6	30	24
1.0	cExp1_6ws	31	20-25	cExp2_d=7	54	54
2.0	cExp1_6ws	6	4-14	cExp2_d=7	36	29
1.0	cExp1_7ws	36	23-29	cExp2_d=8	63	63
2.0	cExp1_7ws	6	4-16	cExp2_d=8	42	34
1.0	cExp1_8ws	41	26-33	cExp2_d=9	72	72
2.0	cExp1_8ws	6	4-18	cExp2_d=9	48	39
1.0	cExp1_9ws	46	29-37	cExp2_d=10	81	81
2.0	cExp1_9ws	6	4-20	cExp2_d=10	54	44
1.0	cExp1_10ws	51	32-41	cExp2_d=11	90	90
2.0	cExp1_10ws	6	4-22	cExp2_d=11	60	49
1.0	cExp1_pws	5*p+1	3*p+2 , 4*p+1	cExp2_d=q	9*(q-1)	9*(q-1)
2.0	cExp1_pws	6	4 , 2*(p+1)	cExp2_d=q	6*(q-1)	5*q-6

Table 2: Comparison between centralized and decentralized derivation methods for algorithm version 1.0 or 2.0.

without consulting the rest of services.

In our second experiment, we use choreographies where only two web services are involved and we modify the *depth* of the tree representing the choreography. Thus, $cExp2\_d = 3$ is a choreography with two branches and depth 3, choreography $cExp2\_d = 4$ is similar to the previous one but with depth 4, and so on up to $cExp2\_d = 11$ with depth 11. Results are also presented in Table 2. As we can observe, in both versions, the number of messages increases linearly with the depth of the choreography. However the number of messages in the optimized version is lower, and it also grows more slowly. This difference would be greater in choreographies with higher number of involved services, because more unnecessary messages would be avoided in optimized derivations. The single branching point of choreographies involves a single service so, for each choreography, all executions of the optimized decentralized derivation require a constant number of messages.

5 Conclusions

In this paper we have introduced a new version of our services derivation tool, DIEGO. To the best of our knowledge, DIEGO is the only tool that derives correct sets of WS-BPEL services from WS-CDL choreographies even in cases where the natural projection does not work, without requiring any well-formedness conditions, and in an integrated way not requiring any human interaction in the derivation. The new version described in this paper, DIEGO 2.0, provides algorithms that are more efficient, and it incorporates additional features such as testing of derived services, which automatically compares the behavior of the derived services with the corresponding choreogra-

phy. Besides, we have formally defined the new derivation algorithms and proved their correctness.

As future work, we plan to enrich our formal FSM-based choreography and orchestration languages with variables and temporal conditions. Moreover, we are planning to extend our operational semantics with the parallel operator and with WS-CDL workunits to model conditional behavior. We will also design new derivation algorithms for these new enhanced models and implement them in DIEGO.

References

References

[1] M. Baldoni, C. Baroglio, A. K. Chopra, N. Desai, V. Patti, and M. P. Singh. Choice, interoperability, and conformance in interaction protocols and service choreographies. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS'09), vol. 2, pp. 843-850*, 2009.

[2] M. Bravetti and G. Zavattaro. Contract-based discovery and composition of web services. In *SFM 2009, Lecture Notes in Computer Science, vol. 5569, pp. 261-295*, 2009.

[3] L. Caires and H. T. Vieira. Conversation types. *Theoretical Computer Science, vol. 411, pp. 4399-4440*, 2010.

[4] G. Decker, O. Kopp, F. Leymann, K. Pfitzner, and M. Weske. Modeling service choreographies using BPMN and BPEL4Chor. In *International Conference on Advanced Information Systems Engineering (CAiSE'08), pp. 79-93*, 2008.

[5] G. Díaz and I. Rodríguez. Automatically deriving choreography-conforming systems of services. In *IEEE International Conference on Services Computing (SCC'09), pp. 9-16*, 2009.

[6] G. Díaz and I. Rodríguez. Checking the conformance of orchestrations with respect to choreographies in web services: A formal approach. In *IFIP International Conference FMOODS and FORTE (FMOODS/FORTE'09), pp. 231-236*, 2009.

[7] R. Khadka, B. Sapkota, L. Ferreira Pires, M. S. van, and S. Jansen. Wscdl to ws-bpel: A case study of atl-based transformation. In *International Workshop on Model Transformation with ATL (MtATL'11), vol. 742, pp. 89-103*, 2011.

[8] I. Lanese, C. Guidi, F. Montesi, and G. Zavattaro. Bridging the gap between interaction- and process-oriented choreographies. In *International Conference on Software Engineering and Formal Methods (SEFM'08), pp. 323-332*, 2008.

[9] R. Lucchi and M. Mazzara. A pi-calculus based semantics for ws-bpel. *Journal of Logic and Algebraic Programming, vol. 70, pp. 96-118*, 2007.

[10] J. Mendling and M. Hafner. From inter-organizational workflows to process execution: Generating bpel from ws-cdl. In *On the Move to Meaningful Internet Systems 2005 (OTM'05 Workshops), vol. 3762, pp. 506-515*. 2005.

[11] Z. Qiu, X. Zhao, C. Cai, and H. Yang. Towards the theoretical foundation of choreography. In *World Wide Web Conference (WWW'07), pp. 973-982*, 2007.

[12] P. Rabanal, J. Mateo, I. Rodriguez, and G. Diaz. DIEGO: A tool for DerIving chorEoGraphy-cOnforming web service systems. In *International Conference on Web Services (ICWS'11), pp. 187-194*, 2011.

[13] I. Rodriguez, G. Diaz, P. Rabanal, and J. A. Mateo. A centralized and a decentralized method to automatically derive choreography-conforming web service systems. *Journal of Logic and Algebraic Programming, vol. 81, pp. 127-159*, 2012.

[14] G. Salaün and T. Bultan. Realizability of choreographies using process algebra encodings. In *International Conference Integrated Formal Methods (IFM'09), pp. 167-182*, 2009.

[15] W. M. P. van der Aalst, M. Dumas, C. Ouyang, A. Rozinat, and H. M. W. Verbeek. Choreography conformance checking: An approach based on BPEL and Petri Nets. In *The Role of Business Processes in Service Oriented Architectures*, 2006.

[16] W. M. P. van der Aalst, N. Lohmann, P. Massuthe, C. Stahl, and K. Wolf. From public views to private views - correctness-by-design for services. In *Web Services and Formal Methods (WS-FM'07), pp. 139,153*, 2007.

Appendix A: Proof of Theorem 3.3

Let us recall that, as mentioned in the bulk of the paper, the formal definitions of conformance relations $\texttt{conf}'_s$, $\texttt{conf}'_p$, $\texttt{conf}'$, $\texttt{conf}^{f}_s$, $\texttt{conf}^{f}_p$, $\texttt{conf}^{f}$ can be found in [13].

Let us analyze step by step the behavior of the system $\mathcal{S}$, checking that (a) all the behaviors it can produce, regardless of whether we take sending or processing events, can be produced by $\mathcal{C}$ (which would show that the relations $\texttt{conf}'_s$, $\texttt{conf}'_p$, $\texttt{conf}'$ hold between $\mathcal{S}$ and $\mathcal{C}$); and (b) that *all* behaviors allowed by $\mathcal{C}$ can be produced indeed by $\mathcal{S}$ (which would imply that we also have the relations $\texttt{conf}^{f}_s$, $\texttt{conf}^{f}_p$, $\texttt{conf}^{f}$). Next we analyze how the system executes each choreography transition, following some transition available at some state of the choreography. We distinguish the following moments in the simulation of the choreography transition $t = s^q \xrightarrow{m/(snd \to adr)} s'^q$, where we consider $t = t_{s^q,snd,j}$:

(1) Initially, the orchestrator orc is at state s^q, service snd is at some state $s^r_{iampassive}$ such that $s^q \in \texttt{upToSnd}(\mathcal{C}, s^r, snd)$, and service adr is at some state $s^p_{iampassive}$ such that we have $s^q \in \texttt{upToRcv}(\mathcal{C}, s^p, adr)$. Besides, the input buffers of all services are empty, and the input buffer of the orchestrator is empty too. Moreover, we also have $D = \emptyset$.

(2) From moment (1), the system will eventually reach a configuration where the orchestrator sends a message $takeyourchoice_j of_{snd} at_{s^q}$ to service snd. This occurs when the orchestrator chooses to trigger transition $t = s^q \xrightarrow{m/(snd \to adr)} s'^q$ (let us note that the orchestrator does not need any message from any other service to take this transition). Let moment (2) denote the execution point where this has just happened. Since the message has just been sent, we have $D = \{(orc, takeyourchoice_j at_{s^q}, snd)\}$. The input buffers of all services are empty, the orchestrator is currently at state $s^q_{startingchoice_j of_{snd}}$, and all the rest of services (including snd) are at their same states as in moment (1).

(3) From moment (2), the system will eventually reach a configuration where the service snd receives the message $takeyourchoice_j at_{s^q}$ at its input buffer, and next snd takes the (only) transition this message allows to trigger from its state $s^r_{iampassive}$ (let us note that, by the construction of the services, service snd is ready, at state $s^r_{iampassive}$, to receive a message $takeyourchoice_j at_{s^f}$ for all state $s^f \in \texttt{upToSnd}(\mathcal{C}, s^r, snd)$). Let moment (3) denote the moment where that transition has just been taken. Due to the construction of services from $\mathcal{C}$, when snd takes this transition, it sends the message denoted by the choreography at its current state (that is, it sends m to adr). So, $(snd, m, adr) \in D$. Note that orc and adr might have evolved from moment (2) to moment (3). In particular, orc might have already sent a message $takechoice_j of_{snd} at_{s^q}$ to adr (so, orc would be at state $s^q_{takingchoice_j of_{snd}}$ instead of $s^q_{startingchoice_j of_{snd}}$) and adr might have already received that message at its input buffer and processed it (so it would be at state $s^q_{takingchoice_j of_{snd} at_{s^q}}$ instead of $s^p_{iampassive}$).

(4) From moment (3), regardless of whether services orc and adr have already evolved as explained at the end of the previous item or not, the system will eventually execute the adr transition where adr receives message m from snd and, at the same transition, it sends message $received$ to orc. Let moment (4) denote this moment. At this point, both snd and adr will be at a state $s'^{q}_{iampassive}$, representing that these services are now passive at the state s'^{q} reached in the choreography by taking the selected choreography transition. Besides, $D = \{(adr, received, orc)\}$.

(5) From moment (4), the orchestrator will eventually receive the message $received$ from adr and reach state s'^{q}. Let moment (5) be this moment. At this point, both snd and adr are a state $s'^{q}_{iampassive}$ and the rest of services are at the same states they were at moment (1). Besides, we have $D = \emptyset$ and the input buffers of all services are empty.

At moment (5), the system fulfills the same conditions as in moment (1) regarding input buffers and the state of the set D, though services orc, snd, and adr are at states s'^{q}, $s'^{q}_{iampassive}$, and $s'^{q}_{iampassive}$, respectively. It is easy to prove, by induction over the number of choreography transitions taken, that after executing any number of choreography transitions and reaching moment (5), the system will necessarily reach moments (2), (3), (4), and (5) as described above for the next choreography transition, which proves property (a) stated at the beginning of this proof. Note that choreography messages are sent at (3) and processed at (4), and moments (3) of *different* choreography transitions (respectively, moments (4)) cannot be mixed up. Moreover, the system can evolve to take a new choreography transition for *any* choreography transition that can be taken from the previous choreography state, which proves property (b) too. Thus, the system of services performs all transitions defined by the choreography. Therefore, the system of services conforms to the choreography with respect to all proposed relations.

Appendix B: Proof of Theorem 3.5

Again, we analyze step by step the behavior of the system $\mathcal{S}$, checking that (a) all the sequences of events it can produce, regardless of whether we take sending or processing events, can be produced by $\mathcal{C}$ (so $\texttt{conf}'_s$, $\texttt{conf}'_p$, $\texttt{conf}'$ hold between $\mathcal{S}$ and $\mathcal{C}$); and (b) *all* behaviors allowed by $\mathcal{C}$ can be produced indeed by $\mathcal{S}$ (so we also have $\texttt{conf}^{f\prime}_s$, $\texttt{conf}^{f\prime}_p$, $\texttt{conf}^{f\prime}$). As we did before, we analyze how the system executes each choreography transition, following some transition available at some state of the choreography. We distinguish the following moments in the simulation of the choreography transition $t = s^q \xrightarrow{m/(snd \to adr)} s'^{q}$, where we consider $t = t_{s^q,snd,j}$:

(1) Initially, the first service with the capability of choosing at the current choreography state, that is $\alpha_1^{s^q}$, is at state $s^q_{icanchoose}$. For any other service id with that capability, i.e. $id \in \{\alpha_2^{s^q}, \ldots, \alpha_{h_{s^q}}^{s^q}\}$, id is at some state $s^r_{iampassive}$ such that $s^q \in \texttt{upToSnd}(\mathcal{C}, s^r, id)$. Let us note that $snd \in \{\alpha_1^{s^q}, \ldots, \alpha_{h_{s^q}}^{s^q}\}$. Besides, service adr is at some state $s^p_{iampassive}$ such that we have $s^q \in \texttt{upToRcv}(\mathcal{C}, s^p, adr)$. The input buffers of all services are empty. Moreover, we also have $D = \emptyset$.

(2) From moment (1), the system will eventually reach a configuration where service snd receives the message $youcanchooseat_{s^q}$ from the previous service in the decision-making sequence, which makes snd reach state $s^q_{icanchoose}$. Let moment (2) denote the moment where this just has happened. All services being before snd in the decision-making sequence have already received the same message from the service before them, so they have moved to state $s^q_{icanchoose}$. However, next they have refused to choose, passing the $youcanchooseat_{s^q}$ message to the next service and reaching state $s^q_{iampassive}$. The rest of services are at the same states as in moment (1). The input buffers of all services are empty and we have $D = \emptyset$ again.

(3) From moment (2), the system will eventually reach a configuration where the service snd sends a message $takemychoice_j at_{s^q}$ to adr and next it sends m to the same service. Let moment (3) denote this execution point. Service snd is at state $s'^q_{iampassive}$ and we have $(snd, m, adr) \in D$. Besides, the service adr might have already processed message $takemychoice_j at_{s^q}$ from snd, or it might have not. If it has, then service adr will be at state $s^q_{takingchoice_j of_{snd}}$ (instead of $s^p_{iampassive}$) and we would have $D = \{(snd, m, adr)\}$ (instead of $D = \{(snd, takemychoice_j at_{s^q}, adr), (snd, m, adr)\}$).

(4) From moment (3), regardless of whether service adr has already evolved as explained at the end of the previous item or not, the system will eventually reach a configuration where service adr processes message m from snd. Let moment (4) denote this point. When adr processes message m, it reaches state $s'^q_{iampassive}$ and sends message $youcanchooseat_{s'^q}$ to the service beginning the decision-making sequence of the new state, that is $\alpha_1^{s'^q}$. Thus, we have $D = \{(adr, youcanchooseat_{s'^q}, \alpha_1^{s'^q})\}$. Besides, the input buffers of all services are empty.

At moment (4), the system fulfills the same conditions as in moment (1) regarding input buffers and the state of the set D, though services snd, and adr are at states $s'^q_{iampassive}$, and $s'^q_{iampassive}$, respectively, and the only message stored in D allows to begin the decision-making sequence of the *next* choreography state s'^q. Besides, all services being before snd in the decision-making sequence of s^q are at state $s^q_{impassive}$, as they were at moment (1) indeed. Again it is easy to prove, by induction over the number of choreography transitions taken, that after executing any number of choreography transitions and reaching moment (4), the system will necessarily reach moments (2), (3), and (4), as described above for the next choreography transition, which proves property (a) stated at the beginning of this proof. Note that choreography messages are sent at (3) and processed at (4), and moments (3) of *different* choreography transitions (respectively, moments (4)) cannot be mixed up. Moreover, the system can evolve to take a new choreography transition for *any* choreography transition that can be taken from the previous choreography state, which proves property (b) too. Thus, the system of services performs all transitions defined by the choreography. Therefore, the system of services conforms to the choreography with respect to all proposed relations.