

Automatically deriving choreography-conforming systems of services

Gregorio Díaz

Dept. de Sistemas Informáticos
Universidad de Castilla-La Mancha
02071. Albacete, Spain
gregorio.diaz@uclm.es

Ismael Rodríguez

Dept. Sist. Informáticos y Computación
Universidad Complutense de Madrid
28040. Madrid, Spain
isrodrig@sip.ucm.es

Abstract

We present a formal method to derive a set of web services from a given choreography, in such a way that the system consisting of these services necessarily conforms to the choreography. A formal model to represent orchestrations and choreographies is given, and we define several conformance semantic relations allowing to detect whether a set of orchestration models representing some web services leads to the overall communications described in a choreography.

1 Introduction

Web services related technologies are a set of middleware technologies for supporting *Service-Oriented Computing* [13]. 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 [1] (*Business Processes for Web Services*), which is a language for describing the web service behavior (workflow) in terms of the composition of other web services. On the other hand, the choreography concerns the *observable* interaction among web services. It can be defined, e.g., by using WS-CDL [18] (*Choreography Description Language for Web Services*). Roughly speaking, the relation between orchestration and choreography can be stated as follows: 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 some formal frameworks to automatically derive web services (in particular, their orchestration definition) from a given choreography, in such a way that the concurrent behavior of these derived services necessarily *conforms* to the choreography. The first derivation method is based on adding an *orchestrator* service, which is a kind of director that is responsible of coordinating services and controlling the system workflow. An alternative method deriving a decentralized system, with no orchestrator, is presented too. In order to fix the meaning of *conformance* in this context, we define several semantic relations such that, given the orches-

tration of some web services and a choreography defining how these web services should interact, they decide whether the interaction of these web services necessarily leads to the required observable behavior. Models of orchestrations and choreographies are constructed by means of two different formal languages. Languages explicitly consider characteristics such as service identifiers, specific senders/addressees, message buffers for representing *asynchronous* communications, or message types.

This paper makes the following contributions. First, the proposed method to derive a conforming set of service models from a given choreography model can be used to define models and early prototypes of web services systems, as well as to formally/empirically analyze the properties of these models/prototypes. Moreover, if service orchestrations do not have to be automatically derived but are *given*, then the proposed conformance relations between orchestrations and choreographies also allow developers to select the adequate service that accomplishes the behavior of certain role, thus aiding web service discovery tasks. Models defined in the proposed modeling languages can be used to analyze the properties of systems of services, such as stuck-freeness and other problems derived from concurrent execution. For instance, by analyzing the order of exchanged messages we can study whether the information is ready when required, which concerns correlation and compensation issues.

2 Related Work

There are few related works that deal with the asynchronous communication in contracts for web service context. In fact, we are only aware of three works from van der Alst et al. [17], Kohei Honda et al. [12] and, Bravetti and Zavattaro [4]. In particular, van der Alst et al. [17] 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. The authors show how the contract refinement can be done independently, and they check whether contracts do not contain cycles. Kohei Honda et al. [12] present a generalization of binary session types to multiparty sessions for π -calculus. They provide a new notion

of types which can directly abstract the intended conversation structure among n -parties as *global scenarios*, retaining an intuitive type syntax. They also provide a consistency criteria for a conversation structure with respect to the protocol specification (contract), and a type discipline for individual processes by using a *projection*. Bravetti and Zavattaro [4] allow to compare systems of orchestrations and choreographies by means of the *testing* relation given by [2, 9]. Systems are represented by using a process algebraic notation, and operational semantics for this language are defined in terms of labeled transitions systems. On the contrary, our framework uses an extension of *finite state machines* to define orchestrations and choreographies, and a semantic relation based on the *conformance* relation [15, 16] is used to compare both models. In addition, let us note that [4] considers the suitability of a service for a given choreography *regardless* of the actual definition of the rest of services it will interact with, i.e. the service must be valid for the considered role *by its own*. This eases the task of finding a suitable service fitting into a choreography role: Since the rest of services do not have to be considered, we can search for suitable services for each role *in parallel*. However, let us note that sometimes this is not realistic. In some situations, the suitability of a service actually depends on the activities provided by the rest of services. For instance, let us consider that a *travel agency* service requires that either the *air company* service or the *hotel* service (or both) provide a transfer to take the client from the airport to the hotel. A hotel providing a transfer is *good* regardless of whether the air company provides a transfer as well or not. However, a hotel not providing a transfer is valid for the travel agency *only if* the air company does provide the transfer. This kind of subtle requirements and conditional dependencies is explicitly considered in our framework. Thus, contrarily to [4], our framework considers that the suitability of a service depends on what the rest of services actually do. Furthermore, this paper presents a method to automatically *derive* services from a choreography in such a way that the system consisting of these services necessarily *conforms* to the choreography. This contrasts with the projection notion given in [4], which does not guarantee that derived services do so.

Other works concern the projection and conformance validation between choreography and orchestration with *synchronous* communication. Bravetti and Zavattaro [3] propose a theory of contracts for conformance checking. They define an effective procedure that can be used to verify whether a service with a given contract can correctly play a specific role within a choreography. In [14], Zongyan et al. define the concept of restricted natural choreography that is easily implementable, and they propose two structural conditions as a criterion to distinguish the restricted natural choreography. Furthermore, they propose a new concept, the *dominant role* of a choice for projection concerns. Carbone et al. [7] study the description of communication behaviors from a global point of view of the communication and end-point behavior levels. Three definitions for proper-structured global description and a theory for projection are developed. Bultan

and Fu [6, 5] specify Web Services as conversations by Finite State Machines that analyze whether UML collaboration diagrams are realizable or not.

3 Formal model

In this section we present our languages to define models of orchestrations and choreographies. This section constitutes an extended, revised, and motivated version of [10], where a brief introduction to these languages is given. Some preliminary notation is presented next.

Definition 3.1 Given a type A and $a_1, \dots, a_n \in A$ with $n \geq 0$, we denote by $[a_1, \dots, a_n]$ the *list* of elements a_1, \dots, a_n of A . We denote the empty list by $[\]$.

Given two lists $\sigma = [a_1, \dots, a_n]$ and $\sigma' = [b_1, \dots, b_m]$ of elements of type A and some $a \in A$, we consider $\sigma \cdot a = [a_1, \dots, a_n, a]$ and $\sigma \cdot \sigma' = [a_1, \dots, a_n, b_1, \dots, b_m]$.

Given a set of lists L , a *path-closure* of L is any subset $V \subseteq L$ such that for all $\sigma \in V$ we have that (a) either $\sigma = [\]$ or $\sigma = \sigma' \cdot a$ for some σ' with $\sigma' \in V$; and (b) there do not exist $\sigma', \sigma'' \in V$ such that $\sigma \cdot a = \sigma'$ and $\sigma \cdot b = \sigma''$ with $a \neq b$.

We say that a path-closure V of L is *complete* in L if it is *maximal* in L , that is, if there does not exist a path-closure $V' \subseteq L$ such that $V \subset V'$. The set of all complete path-closures of L is denoted by $\text{Comp}(L)$. \square

We present our model of web service *orchestration*. The internal behavior of a web service in terms of its interaction with other web services is represented by a *finite state machine* where, at each state s , the machine can receive an input i and produce an output o as response before moving to a new state s' . Moreover, each transition explicitly defines which service must send i : A sender identifier *snd* is attached to the transition denoting that, if i is sent by service *snd*, then the transition can be triggered. We assume that all web services are identified by a given identifier belonging to a set ID . Moreover, transitions also denote the *addressee* of the output o , which is denoted by an identifier *adr*. Let us note that web services receive messages asynchronously. This is represented in the model by considering an *input buffer* where all inputs received and not processed yet are cumulated. Each input has attached the identifier of the sender of the input. A *partition* of the set of possible inputs will be explicitly provided, and each set of the partition will denote a *type of inputs*. If a service transition requires receiving an input i whose type is t , then we will check if the first message of type t appearing in the input buffer is i indeed. If it is so (the predicate `available` given in the next definition will be used to check this), then we will be able to consume the input from the input buffer and take the transition.¹

Definition 3.2 Given a set of service identifiers ID , a *service* for ID is a tuple $(id, S, I, O, s_{in}, T, \psi)$ where $id \in ID$ is the

¹Note that, equivalently, we could speak about *different* input buffers, one for each type, rather than a single input buffer.

identifier of the service, S is the set of states, I is the set of inputs, O is the set of outputs, $s_{in} \in S$ is the initial state, T is the set of transitions, and ψ is a partition of I , i.e. we have $\bigcup_{p \in \psi} p = I$ and for all $p, p' \in \psi$ we have $p \cap p' = \emptyset$. 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 an input, $snd \in ID$ is the required sender of i , $o \in O$ is an output, and $adr \in ID$ is the addressee of o . A transition (s, i, snd, o, adr, s') is also denoted by $s \xrightarrow{(snd,i)/(adr,o)} s'$.

Given a service $M = (id, S, I, O, s_{in}, T)$, an *input buffer* for M is a list $[(id_1, i_1), \dots, (id_k, i_k)]$ where $id_1, \dots, id_k \in ID$ and $i_1, \dots, i_k \in I$. A *configuration* of M is a pair $c = (s, b)$ where $s \in S$ is a state of M and b is an input buffer for M . The set of all input buffers is denoted by \mathcal{B} . The *initial configuration* of M is $(s_{in}, [])$.

Let us suppose that, given a set S , 2^S denotes the powerset of S . Let $b = [(id_1, i_1), \dots, (id_k, i_k)] \in \mathcal{B}$ with $k \geq 0$ be an input buffer, $id \in ID$, $i \in I$, and $S \in 2^I$. We have $\text{available}(b, id, i, S)$ iff, for some $1 \leq j \leq k$, we have $(id_j, i_j) = (id, i)$ and there do not exist $l < j$, $id' \in ID$, and $i' \in S$, such that $(id_l, i_l) = (id', i')$. We have $\text{insert}(b, id, i) = b \cdot (id, i)$. In addition, we also have $\text{remove}(b, id, i) = [(id_1, i_1), \dots, (id_{j-1}, i_{j-1}), (id_{j+1}, i_{j+1}), \dots, (id_k, i_k)]$, provided that $j \in \mathbf{N}$ is the minimum value such that $j \in [1..k]$, $id = id_j$, and $i = i_j$. \square

Next we compose services into systems of services.

Definition 3.3 Let $ID = \{id_1, \dots, id_p\}$. For all $1 \leq j \leq p$, let $M_j = (id_j, S_j, I_j, O_j, s_{j,in}, T_j, \psi_j)$ be a service for ID . Then, $\mathcal{S} = (M_1, \dots, M_p)$ is a *system of services* for ID .

For all $1 \leq j \leq p$, let c_j be a configuration of M_j . We say that $c = (c_1, \dots, c_p)$ is a *configuration* of \mathcal{S} . Let c'_1, \dots, c'_p be the initial configurations of M_1, \dots, M_p , respectively. Then, (c'_1, \dots, c'_p) is the *initial configuration* of \mathcal{S} . \square

We formally define how systems *evolve*, i.e. how a service of the system triggers a transition and how this affects other services in the system. Outputs of services will be considered as inputs of the services these outputs are sent to. Besides, we consider a special case of input/output that will be used to denote a *null* communication. If the input of a transition is *null* then we are denoting that the service can take this transition without waiting for any previous message from any other service, that is, we denote a *proactive* action of the service. Similarly, a *null* output denotes that no message is sent to other service after taking the corresponding transition. In both cases, the sender and the addressee of the transition are irrelevant, respectively, so in these cases they will also be denoted by a *null* symbol. A system evolution will be denoted by a tuple $(c, snd, i, proc, o, adr, c')$ where c and c' are the initial and the final configuration of the system, respectively, i is the input processed in the evolution, o is the output sent as result of the evolution, $proc$ is the service whose transition is taken in the evolution, snd is the sender of i , and adr is

the addressee of o . There are two reasons why an evolution can be produced: (a) a service proactively initiates a transition, that is, a transition whose input is *null* is taken; and (b) a service triggers a transition because there is an available message in its input buffer labelled by the sender identifier and the input required by the transition. In both cases (a) and (b), there are two possibilities regarding a new output is sent or not: (1) if the transition denotes a *null* output then no other input buffer is modified; (2) otherwise, i.e. if the transition denotes an output different from *null*, then this output is stored in the buffer of the addressee as an *input*. By considering any combination of either (a) or (b) with either (1) or (2), four kinds of evolutions arise indeed.

Definition 3.4 Let $ID = \{id_1, \dots, id_p\}$ be a set of service identifiers and $\mathcal{S} = (M_1, \dots, M_p)$ be a *system of services* for ID where for all $1 \leq j \leq p$ we have that $M_j = (id_j, S_j, I_j, O_j, s_{j,in}, T_j, \psi_j)$. Let $c = (c_1, \dots, c_p)$ be a configuration of \mathcal{S} where for all $1 \leq j \leq p$ we have $c_j = (s_j, b_j)$.

An *evolution* of \mathcal{S} from the configuration c is any tuple $(c, snd, i, proc, o, adr, c')$ where $i \in I_1 \cup \dots \cup I_p$ is the input of the evolution, $o \in O_1 \cup \dots \cup O_p$ is the output of the evolution, $c' = ((s'_1, b'_1), \dots, (s'_p, b'_p))$ is the new configuration of \mathcal{S} , and $snd, proc, adr \in ID$ are the sender, the processor, and the addressee of the evolution, respectively. All these elements must be defined according to one of the following choices:

- (a) (*evolution activated by some service by itself*) For some $1 \leq j \leq p$, let us suppose $s_j \xrightarrow{(null,null)/(adr',o)} s' \in T_j$. Then, $s'_j = s'$ and $b'_j = b_j$. Besides, $snd = null$, $proc = id_j$, $adr = adr'$;
- (b) (*evolution activated by processing a message from the input buffer of some service*) For some $1 \leq j \leq p$, let us suppose that $s_j \xrightarrow{(snd',i)/(adr',o)} s' \in T_j$ and we have $\text{available}(b_j, snd', i, p)$, where p is the only set belonging to ψ_j such that $i \in p$. Then, $s'_j = s'$ and $b'_j = \text{remove}(b_j, snd', i)$. Besides, $snd = snd'$, $proc = id_j$, and $adr = adr'$;

where, both in (a) and (b), the new configurations of the rest of services are defined according to one of the following choices:

- (1) (*no message is sent to other service*) If $adr' = null$ or $o = null$ then for all $1 \leq q \leq k$ with $q \neq j$ we have $s'_q = s_q$ and $b'_q = b_q$.
- (2) (*a message is sent to other service*) Otherwise, let $id_g = adr'$ for some $1 \leq g \leq k$. Then, we have $s'_g = s_g$ and $b'_g = \text{insert}(b_g, id_j, o)$. Besides, for all $1 \leq q \leq k$ with $q \neq j$ and $q \neq g$ we have $s'_q = s_q$ and $b'_q = b_q$. \square

Figure 1 (left and center) shows a simple client/server orchestration specification where the client (A) sends requests to the server (B) and the server responds to them, until the

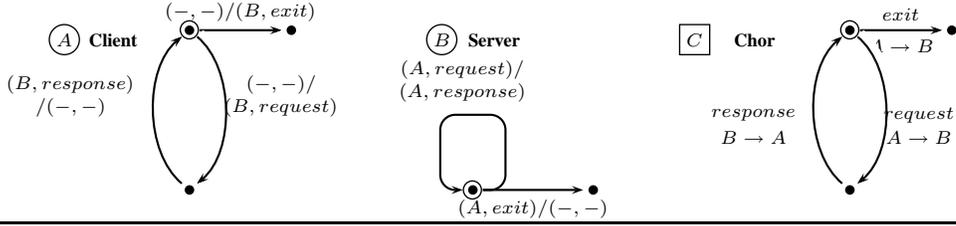


Figure 1. A client/server orchestration (left and center) and a choreography specification (right).

client notifies that it leaves the system. Initial states are denoted by a double circle node, and *null* inputs and outputs are denoted by the dash symbol.

As we will see later, the conformance of a system of service orchestrations with respect to a choreography will be assessed in terms of the behaviors of both machines. We extract the behaviors of systems of services as follows: Given any sequence of consecutive evolutions of the system from its initial configuration, we take the sequence of inputs and outputs labelling each evolution and we remove all *null* elements from this sequence. The extracted sequence (called *trace*) represents the *effective* behavior of the original sequence. We distinguish two kinds of traces. A *sending trace* is a sequence of outputs ordered as they are *sent* by their corresponding senders. A *processing trace* is a sequence of inputs ordered as they are *processed* by the services which receive them, that is, they are ordered as they are taken from the input buffer of each addressee service to trigger some of its transitions. Both traces attach some information to explicitly denote the services involved in each operation.

Definition 3.5 Let \mathcal{S} be a system, c_1 be the initial configuration of \mathcal{S} , and $(c_1, snd_1, i_1, proc_1, o_1, adr_1, c_2), (c_2, snd_2, i_2, proc_2, o_2, adr_2, c_3), \dots, (c_k, snd_k, i_k, proc_k, o_k, adr_k, c_{k+1})$ be k consecutive evolutions of \mathcal{S} .

Let $a_1 \leq \dots \leq a_r$ denote all indexes of non-null outputs in the previous sequence, i.e. we have $j \in \{a_1, \dots, a_r\}$ iff $o_j \neq null$. Then, $[(proc_{a_1}, o_{a_1}, adr_{a_1}), \dots, (proc_{a_r}, o_{a_r}, adr_{a_r})]$ is a *sending trace* of \mathcal{S} . In addition, if there do not exist $snd', i', proc', o', adr', c'$ such that $(c_{k+1}, snd', i', proc', o', adr', c')$ is an evolution of \mathcal{S} then we also say that $[(proc_{a_1}, o_{a_1}, adr_{a_1}), \dots, (proc_{a_r}, o_{a_r}, adr_{a_r}), stop]$ is a *sending trace* of \mathcal{S} . The set of all sending traces of \mathcal{S} is denoted by $sndTraces(\mathcal{S})$.

Let $a_1 \leq \dots \leq a_r$ denote all indexes of non-null inputs in the previous sequence, i.e. we have $j \in \{a_1, \dots, a_r\}$ iff $i_j \neq null$. Then, $[(snd_{a_1}, i_{a_1}, proc_{a_1}), \dots, (snd_{a_r}, i_{a_r}, proc_{a_r})]$ is a *processing trace* of \mathcal{S} . In addition, if there do not exist $snd', i', proc', o', adr', c'$ such that $(c_{k+1}, snd', i', proc', o', adr', c')$ is an evolution of \mathcal{S} then we also say that $[(snd_{a_1}, i_{a_1}, proc_{a_1}), \dots, (snd_{a_r}, i_{a_r}, proc_{a_r}), stop]$ is a *processing trace* of \mathcal{S} . The set of all processing traces of \mathcal{S} is denoted by $procTraces(\mathcal{S})$. \square

Next we introduce our formalism to represent choreographies. Contrarily to systems of orchestrations, this formalism

focuses on representing the interaction of services as a whole. Thus a single machine, instead of the composition of several machines, is considered. Each choreography transition denotes a *message action* where some service sends a message to another one.

Definition 3.6 A *choreography machine* \mathcal{C} is a tuple $\mathcal{C} = (S, M, ID, s_{in}, T)$ 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 $s \xrightarrow{m/(snd \rightarrow adr)} s'$.

A *configuration* of \mathcal{C} is any state $s \in S$. An *evolution* of \mathcal{C} from the configuration $s \in S$ is any transition $(s, m, snd, adr, s') \in T$ from state s . The *initial configuration* of \mathcal{C} is s_{in} . \square

Coming back to our previous example, Figure 1 (right) depicts a choreography \mathcal{C} between services A and B , that is, the client and the server. The transitions of this choreography actually denote the same evolutions we can find in a system of services consisting of services A and B .

As we did before for systems of services, next we identify the sequences of messages that can be produced by a choreography machine.

Definition 3.7 Let c_1 be the initial configuration of a choreography machine \mathcal{C} . Let $(c_1, m_1, snd_1, adr_1, c_2), \dots, (c_k, m_k, snd_k, adr_k, c_{k+1})$ be $k \geq 0$ consecutive evolutions of \mathcal{C} . We say that $\sigma = [(snd_1, m_1, adr_1), \dots, (snd_k, m_k, adr_k)]$ is a *trace* of \mathcal{C} . In addition, if there do not exist m', snd', adr', c' such that $(c_{k+1}, m', snd', adr', c')$ is an evolution of \mathcal{C} then we also say that $[(snd_1, m_1, adr_1), \dots, (snd_k, m_k, adr_k), stop]$ is a *trace* of \mathcal{C} . The set of all traces of \mathcal{C} is denoted by $traces(\mathcal{C})$. \square

4 Conformance relations and derivation of choreography-compliant sets of services

Now we are provided with all the required formal machinery to define our *conformance relations* between systems of

orchestrations and choreographies. We will consider a semantic relation inspired in the *conformance testing* relation given in [15, 16]. This notion is devoted to check whether an *implementation* meets the requirements imposed by a *specification*. In our case, we will check whether the behavior of a system of orchestration services meets the requirement given by the choreography.

However, there are some important differences between the notion proposed in [15, 16] and the notion considered here. Contrarily to those works, the behavior of orchestrations and choreographies will not be compared in terms of their possible interactions with an external entity (i.e. user, observer, external application, etc) but in terms of what both models can/cannot do by their own, because both models are considered as *closed worlds*. Let us also note that non-determinism allows a choreography to provide multiple valid ways to perform the operations it defines. Consequently, we consider that a system of orchestration services conforms to a choreography if it performs *one or more* of these valid ways. For each of these valid ways, care must be taken not to allow the system of services to *incompletely* perform it, i.e. to finish in an intermediate state – provided that the choreography does not allow it either. In order to check these requirements, only complete path-closures will be considered (see Definition 3.1). Moreover, the set of complete path-closures of the system of choreographies is required to be non-empty because the system is required to provide at least *one* (complete) way to perform the requirement given by the choreography. Alternatively, we also consider another relation where the system of orchestrations is required to perform *all* execution ways defined by the choreography. This alternative notion will be called *full conformance*.

Let us recall that we consider asynchronous communications in our framework. Thus, the moment when a message is sent does not necessarily coincide with the moment when this message is taken by the receiver from its input buffer and is processed. In fact, we can define a choreography in such a way that defined communications refer to either the former kind of events or the latter (i.e., instants where messages are sent, or instants where messages are processed by their receivers, respectively). Thus, we consider two ways in which a system of services may conform to a choreography: with respect to sending traces, and with respect to processing traces. Besides, we explicitly identify the case where both conformance notions simultaneously hold.

Definition 4.1 Let \mathcal{S} be a system of services and \mathcal{C} be a choreography machine.

We say that \mathcal{S} *conforms to \mathcal{C} with respect to sending actions*, denoted by $\mathcal{S} \text{ conf}_s \mathcal{C}$, if either we have $\emptyset \subset \text{Comp}(\text{sndTraces}(\mathcal{S})) \subseteq \text{Comp}(\text{traces}(\mathcal{C}))$ or we have $\emptyset = \text{Comp}(\text{sndTraces}(\mathcal{S})) = \text{Comp}(\text{traces}(\mathcal{C}))$.

We say that \mathcal{S} *fully conforms to \mathcal{C} with respect to sending actions*, denoted by $\mathcal{S} \text{ conf}_s^f \mathcal{C}$, if $\text{Comp}(\text{sndTraces}(\mathcal{S})) = \text{Comp}(\text{traces}(\mathcal{C}))$.

We say that \mathcal{S} *conforms to \mathcal{C} with respect to processing actions*, denoted by $\mathcal{S} \text{ conf}_p \mathcal{C}$, if we have either

$$\emptyset \subset \text{Comp}(\text{prcTraces}(\mathcal{S})) \subseteq \text{Comp}(\text{traces}(\mathcal{C})) \text{ or } \emptyset = \text{Comp}(\text{prcTraces}(\mathcal{S})) = \text{Comp}(\text{traces}(\mathcal{C})).$$

We say that \mathcal{S} *fully conforms to \mathcal{C} with respect to sending actions*, denoted by $\mathcal{S} \text{ conf}_p^f \mathcal{C}$, if $\text{Comp}(\text{prcTraces}(\mathcal{S})) = \text{Comp}(\text{traces}(\mathcal{C}))$.

We say that \mathcal{S} *conforms to \mathcal{C}* , denoted by $\mathcal{S} \text{ conf } \mathcal{C}$, if $\mathcal{S} \text{ conf}_s \mathcal{C}$ and $\mathcal{S} \text{ conf}_p \mathcal{C}$.

We say that \mathcal{S} *fully conforms to \mathcal{C}* , denoted by $\mathcal{S} \text{ conf}^f \mathcal{C}$, if $\mathcal{S} \text{ conf}_s^f \mathcal{C}$ and $\mathcal{S} \text{ conf}_p^f \mathcal{C}$. \square

The subtle differences between all the previous semantic relations are illustrated in detail, by means of several examples and a small case study, in Appendix A.

Once we are provided with appropriate notions to compare sets of orchestration models with choreography models, we study the problem of automatically deriving orchestration services from a given choreography, in such a way that the system consisting of these derived services conforms to the choreography. Next we consider deriving services by *projecting* the structure of the choreography into each involved service. Each service copies the form of states and transitions of the choreography, though service transitions are labeled only by actions concerning the service. Unfortunately, if services are derived in this way then, in general, the resulting set of services does not conform to the choreography with respect to any of the proposed conformance notions. Let us consider Figure 4, depicted in Appendix A. Services 24, 25, and 26 are projections of the choreography 27 into each service regarded in the definition of 27. However, the composition of 24, 25, and 26 does not necessarily lead to the behavior required by 27. In particular, it could be the case that service 25 takes its right choice (i.e. it sends c to 26) while 24 takes its left choice (i.e. it sends b to 25), which is not allowed by 27. Moreover, if *only* messages appearing in choreography 27 are allowed in services then no alternative definition of 24, 25, and 26 allows to meet the requirement imposed by 27: Service 24 cannot decide whether it must send b or c to 25 because it cannot know the message sent by 25 to 26. The problem of investigating how we can design asynchronous communicating processes in such a way they will necessarily produce some behavior or reach some configuration have been tackled in several ways in the literature. For instance, [11] studies the problem of designing two asynchronous processes in such a way that their progress is guaranteed, while [8] studies the cases where we cannot define some communicating processes conforming to a given specification. We will make any choreography realizable by *adding* some control messages to the definition of services. These messages will allow services to know what is required at each time to properly make the next decision, according to the choreography specification. Next we reconsider our conformance relations under the assumption that these additional messages are allowed indeed. That is, services are allowed to send/receive additional messages not included in the choreography. In order to avoid confusion between standard choreography messages and other messages, the latter messages are required to be different to the former. Regarding the definition of conformance relations, we require

traces inclusion/equality again, though we remove additional messages prior to comparing sets of traces.

Definition 4.2 Let $\sigma \in \text{sndTraces}(\mathcal{S}) \cup \text{prcTraces}(\mathcal{S})$ where \mathcal{S} is a system of services. The *constrain* of σ to a set of inputs and outputs Q , denoted by σ^Q , is the result of removing from σ all elements (a, m, b) with $m \notin Q$.

Let \mathcal{S} be a system of services for ID and let $\mathcal{C} = (S, M, ID, s_{in}, T)$ be a choreography. Let $\text{conf}_x \in \{\text{conf}_s, \text{conf}_s^f, \text{conf}_p, \text{conf}_p^f\}$. We have $\mathcal{S} \text{ conf}'_x \mathcal{C}$ if $\mathcal{S} \text{ conf}_x \mathcal{C}$ provided that the occurrences of $\text{sndTraces}(\mathcal{S})$ and $\text{prcTraces}(\mathcal{S})$ appearing in Definition 4.1 are replaced by sets $\{\sigma^M \mid \sigma \in \text{sndTraces}(\mathcal{S})\}$ and $\{\sigma^M \mid \sigma \in \text{prcTraces}(\mathcal{S})\}$, respectively. Now, let $\text{conf}_x \in \{\text{conf}, \text{conf}^f\}$. We have $\mathcal{S} \text{ conf}'_x \mathcal{C}$ if $\mathcal{S} \text{ conf}_x \mathcal{C}$ provided that the occurrences of $\text{conf}_s, \text{conf}_s^f, \text{conf}_p, \text{conf}_p^f$ appearing in the definition of conf and conf^f , given in Definition 4.1, are replaced by $\text{conf}'_s, \text{conf}'_s^f, \text{conf}'_p, \text{conf}'_p^f$, respectively. \square

We revisit our previous example. Let us modify services 24 and 25 in such a way that, right after 25 sends b or c to service 26, service 25 tells service 24 whether b or c was sent. This is done by sending to service 24 a new message d or e , respectively. Services 24' and 25' (also depicted in Figure 4) are the resulting new versions of services 24 and 25, respectively. Let us note that the system consisting in 24', 25', and 26 conforms to 27 with respect to all conformance relations introduced in the previous definition, because all of them ignore messages d and e .

Intuitively, a derivation of services based on a simple projection does not work because it does not make services follow the non-deterministic choices taken by the choreography. In order to solve this problem, next we consider an alternative way to extract services from the choreography. In particular, new control messages are added to make all services follow the same non-determinism choices of the choreography, as we did in our previous example. In order to do it, we will introduce a new service, called the *orchestrator*, which will be responsible of making all non-deterministic choices of the choreography. For each state s_j of the choreography having several outgoing transitions, an equivalent transition will be non-deterministically taken by the orchestrator (say, the p -th available transition). Next, the orchestrator will take several consecutive transitions to *announce* its choice to all services. In each of these transitions, the orchestrator will send a message a_{jp} to another service, meaning that the p -th transition leaving state s_j must be taken by the service. After (a) the orchestrator announces its choice to all services; and (b) the orchestrator receives a message b_{jp} from the *addressee* of the choreography transition (this message denotes that the addressee has processed the message), the orchestrator will reach a state representing the state reached in the choreography after taking the selected transition, and the same process will be followed again. By adding the orchestrator, we make sure that all services follow the same non-deterministic choices of the choreography, and thus a system consisting of

the orchestration and the corresponding derived services will *conform* to the choreography with respect to all conf'_x relations given in Definition 4.2. Let us note that, since the only message required by the orchestrator to continue is sent by the addressee denoted in the choreography transition, at a given time the orchestrator and the services could have reached different steps of the choreography simulation execution (in general, the orchestrator will be in a *further* step). There is no risk that services confuse the order in which each transition must be taken, because all messages controlling transition choices are introduced in input buffers (as the rest of messages) and they will belong to the same *type*. Thus, they will be processed in the same order as the orchestrator sent each of them. This guarantees that services will be led through the choreography graph by following the orchestrator plan, in the same order as planned. Next we will assume that the identifier of the orchestrator is *orc*.

Definition 4.3 Let $\mathcal{C} = (S, M, ID, s_{in}, T)$ be a choreography machine where $ID = \{id_1, \dots, id_n\}$ and $S = \{s_1, \dots, s_l\}$. For all $1 \leq i \leq n$, the *controlled service* for \mathcal{C} and id_i , denoted $\text{controlled}(\mathcal{C}, id_i)$, is a service

$$M_i = \left(\begin{array}{l} id_i, S \cup \{s_{ij}, s'_{ij} \mid i, j \in [1..l]\}, \\ M \cup \{a_{ij} \mid i, j \in [1..l]\}, M \cup \{b_{ij} \mid i, j \in [1..l]\}, \\ s_{in}, T_i, \{\{m\} \mid m \in M\} \cup \{\{a_{ij} \mid i, j \in [1..l]\}\} \end{array} \right)$$

where for all $s_j \in S$ the following transitions are in T_i :

- Let t_1, \dots, t_k be the transitions leaving s_j in \mathcal{C} . For all $1 \leq p \leq k$ we have $s_j \xrightarrow{(orc, a_{jp}) / (null, null)} s_{jp} \in T_i$.
- Let $t_p = s_j \xrightarrow{m / (snd \rightarrow adr)} s'_j \in T$ be the p -th transition leaving s_j in \mathcal{C} . For all $1 \leq j \leq l$ and $1 \leq p \leq k$ we have $s_{jp} \xrightarrow{(snd', i) / (adr', o)} u_{jp} \in T_i$, where
 - (a) if $snd = id_i$ then $snd' = i = null, adr' = adr, o = m$, and $u_{jp} = s'_j$.
 - (b) else, if $adr = id_i$ then $snd' = snd, i = m, adr' = o = null$, and $u_{jp} = s'_{jp}$. Besides, we also have $s'_{jp} \xrightarrow{(null, null) / (orc, b_{jp})} s'_j$ in T_i .
 - (c) else $snd' = i = adr' = o = null$ and $u_{jp} = s'_j$.

The *orchestrator* of \mathcal{C} , denoted by $\text{orchestrator}(\mathcal{C})$, is a service

$$O = \left(\begin{array}{l} orc, S \cup \{s_{ijk} \mid i, j \in [1..l], k \in [1..n+1]\}, \\ M \cup \{b_{ij} \mid i, j \in [1..l]\}, M \cup \{a_{ij} \mid i, j \in [1..l]\}, \\ s_{in}, T_o, \{\{m\} \mid m \in M\} \cup \{\{a_{ij} \mid i, j \in [1..l]\}\} \end{array} \right)$$

where for all $s_j \in S$ the following transitions are included in T_o :

- Let t_1, \dots, t_k be the transitions leaving s_j in \mathcal{C} . For all $1 \leq p \leq k$ we have $s_j \xrightarrow{(null, null) / (null, null)} s_{jp1} \in T_o$.

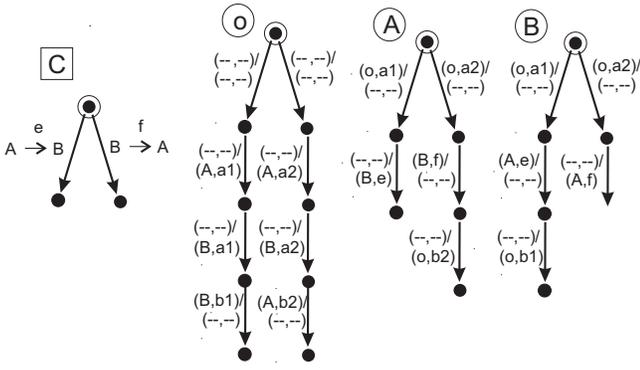


Figure 2. Derivation of services with orchestrator.

- Let $t_p = s_j \xrightarrow{m/(snd \rightarrow adr)} s'_j \in T$ be the p -th transition leaving s_j in \mathcal{C} . For all $1 \leq p \leq k$, $1 \leq i \leq n$ we have $s_{jpi} \xrightarrow{(null, null)/(id_i, a_{jp})} s_{jpi+1} \in T_o$. We also have $s_{jpn+1} \xrightarrow{(adr, b_{jp})/(null, null)} s'_j \in T_o$. \square

Theorem 4.4 Let $\mathcal{C} = (S, M, ID, s_{in}, T)$ be a choreography with $ID = \{id_1, \dots, id_n\}$. Let $\mathcal{S} = (\text{controlled}(\mathcal{C}, id_1), \dots, \text{controlled}(\mathcal{C}, id_n), \text{orchestrator}(\mathcal{C}))$. For all $\text{conf}_x \in \{\text{conf}'_s, \text{conf}^f_p, \text{conf}', \text{conf}^f_s, \text{conf}^f_p, \text{conf}^f\}$ we have $\mathcal{S} \text{ conf}_x \mathcal{C}$. \square

Figure 2 shows a choreography \mathcal{C} as well as the services derived from \mathcal{C} by applying Definition 4.3, including an orchestrator O .

If we do not need to meet the conformance with respect to processing traces, that is, if we only require conf'_s and conf^f_s , then we do not need to require that addressees of choreography transitions *block* the advance of the orchestrator until they process received messages. This restriction was imposed just to force the message processing follow the order required by the choreography. Alternatively, if addressees do not block the orchestrator then, for instance, the service responsible of processing the second message of the execution could process it before the service responsible of processing the first one does so. Even if the orchestrator were not required to wait for the addressees, the order in which messages are *sent* would be correct as long as the orchestrator is required to wait for the *senders*. Actually, if we only consider conformance with respect to sending traces then replacing the restriction of waiting for the addressees by the restriction of waiting for the senders is a good choice in terms of efficiency. This is because, in this case, the orchestrator will not be blocked just waiting for the message to be processed; on the contrary, it will be able to go on even if the message has not been processed yet. Thus, by taking this alternative, the rate of activities the services can actually execute in *parallel* is increased.

Definition 4.5 We have that $\text{controlled}'(\mathcal{C}, id_i)$ is defined as $\text{controlled}(\mathcal{C}, id_i)$ after replacing cases (a) and (b) of Definition 4.3 by the following expressions:

- if $snd = id_i$ then $snd' = i = null$, $adr' = adr$, $o = m$, and $u_{jp} = s'_{jp}$. Besides, we also have $s'_{jp} \xrightarrow{(null, null)/(orc, b_{jp})} s'_j$ in T_i .
- else, if $adr = id_i$ then $snd' = snd$, $i = m$, $adr' = o = null$, and $u_{jp} = s'_j$. \square

Theorem 4.6 Let $\mathcal{C} = (S, M, ID, s_{in}, T)$ be a choreography with $ID = \{id_1, \dots, id_n\}$. Let $\mathcal{S} = (\text{controlled}'(\mathcal{C}, id_1), \dots, \text{controlled}'(\mathcal{C}, id_n), \text{orchestrator}(\mathcal{C}))$. For all $\text{conf}_x \in \{\text{conf}'_s, \text{conf}^f_s\}$ we have $\mathcal{S} \text{ conf}_x \mathcal{C}$. \square

Let us note that we can remove the orchestrator and distribute its responsibilities among the services themselves, thus making a decentralized solution. Let s be a choreography state with several outgoing transitions. Instead of using an orchestrator to choose which transition is taken, we do as follows: We sort all outgoing transitions e.g. by the name of the sender and we make the first sender choose between (a) taking any of the transitions where it is the sender; or (b) refusing to do so. In case (a) it will announce its choice to the rest of services, thus playing the role of the orchestrator in this step. In case (b) it will notify its rejection to choose a transition to the second service. Then, the second service will choose either (a) or (b) in the same way, and so on up to the last sender, which will be forced to take one of its transitions. Let us note that, in this alternative design, a service can receive the request to take a given non-deterministic choice from *several* services, and thus all corresponding transitions must be created. This complicates the definition of the derivation; due to the lack of space, the formal definition of this derivation is given in Appendix B (see Definitions 5.1 and 5.1). As it is shown in Theorems 5.2 and 5.2, the set of services derived in this way also conforms to the choreography with respect to all relations given in Definition 4.2 (if services wait for the addressee of the choreography transition) or with respect to conf'_s and conf^f_s (if they do not).

An example of derivation of the former kind is depicted in Figure 3. For the sake of simplicity, some transitions included in the formal derivation have been omitted. Service A receives the responsibility of either taking one of the transitions where it is the sender (there is only one in this example) or refusing to do so. In the former case, it tells the next service in the list (B) that it will decide the transition indeed (message $a2$) and next it tells all services (i.e. just B) which of its transitions it will actually take ($a21$). Then, it sends e to B and waits for a signal indicating that B has processed the message ($b2$). In the latter case, i.e. if it refuses to choose one of its transitions, then it tells its decision to next service B (message $a1$) and waits for the rest of services (just B) to tell it which choice it must take. When B does so ($a11$), it waits for receiving b from B and next it acknowledges the

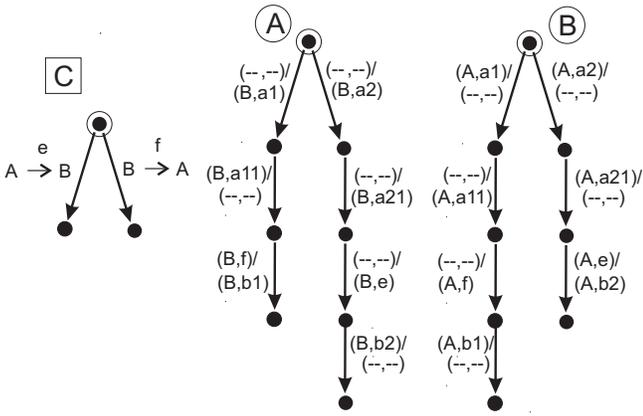


Figure 3. Derivation of services without orchestrator.

reception (b_1). The behavior of B turns out to be dual to the behavior of A .

5 Conclusions and future work

In this paper we have presented a formal framework to automatically extract a system of services that conforms to a given choreography. Two derivation methods, one of them based on an orchestrator service and the other one yielding a decentralized system, are presented. For each method, we consider two alternatives: Making the system conform with respect to instants where messages are sent, or making it conform with respect to all proposed criteria. Languages for defining models of orchestrations and choreographies have been presented, and we have defined some formal semantic relations where, in particular, sending traces are distinguished from processing traces, and the suitability of a service for a given choreography may depend on the activities of the rest of services it will be connected with, which contrasts with previous works [4]. The proposed framework is illustrated with several toy examples and a small case study, given Appendix A.

References

- [1] T. Andrews and F. Curbera. Web Service Business Process Execution Language, Working Draft, 2004. Version 2.0, 1.
- [2] M. Boreale, R. D. Nicola, and R. Pugliese. Trace and testing equivalence on asynchronous processes. *Inf. Comput.*, 172(2):139–164, 2002.
- [3] M. Bravetti and G. Zavattaro. Towards a unifying theory for choreography conformance and contract compliance. In *Software Composition*, pages 34–50, 2007.
- [4] M. Bravetti and G. Zavattaro. Contract compliance and choreography conformance in the presence of message queues. In *Proc. of 5th International workshop on Web Services and Formal Methods, WS-FM’08, LNCS (in press)*. Springer, 2008.
- [5] T. Bultan and X. Fu. Choreography modeling and analysis with collaboration diagrams. *IEEE Data Eng. Bull.*, 31(3):27–30, 2008.
- [6] T. Bultan and X. Fu. Specification of realizable service conversations using collaboration diagrams. *Service Oriented Computing and Applications*, 2(1):27–39, 2008.
- [7] M. Carbone, K. Honda, and N. Yoshida. Theoretical aspects of communication-centred programming. *Electr. Notes Theor. Comput. Sci.*, 209:125–133, 2008.
- [8] H. Castejón, R. Braek, and G. Bochmann. Realizability of collaboration-based service specifications. *Asia-Pacific Software Engineering Conference*, 0:73–80, 2007.
- [9] I. Castellani and M. Hennessy. Testing theories for asynchronous languages. In *FSTTCS*, pages 90–101, 1998.
- [10] G. Díaz and I. Rodríguez. Checking the conformance of orchestrations with respect to choreographies in web services: A formal approach (*to appear as short paper*). In *FMOODS/FORTE 2009*. Springer, 2009.
- [11] M. Gouda and Y. Yu. Synthesis of communicating finite-state machines with guaranteed progress. *IEEE Transactions on Communications*, 32(7):779–788, 1984.
- [12] K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. In *POPL*, pages 273–284, 2008.
- [13] M. Huhns and M. Singh. Service-oriented computing: Key concepts and principles. In *IEEE Internet Computing*, pages 75–81. IEEE Computer Society Press, 2005.
- [14] Z. Qiu, X. Zhao, C. Cai, and H. Yang. Towards the theoretical foundation of choreography. In *WWW*, pages 973–982, 2007.
- [15] J. Tretmans. Conformance testing with labelled transition systems: Implementation relations and test generation. *Computer Networks and ISDN Systems*, 29:49–79, 1996.
- [16] J. Tretmans. Testing concurrent systems: A formal approach. In *CONCUR’99, LNCS 1664*, pages 46–65. Springer, 1999.
- [17] 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 *WS-FM*, pages 139–153, 2007.
- [18] W3C. Web Services Choreography Description Language, 2004. <http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217>.

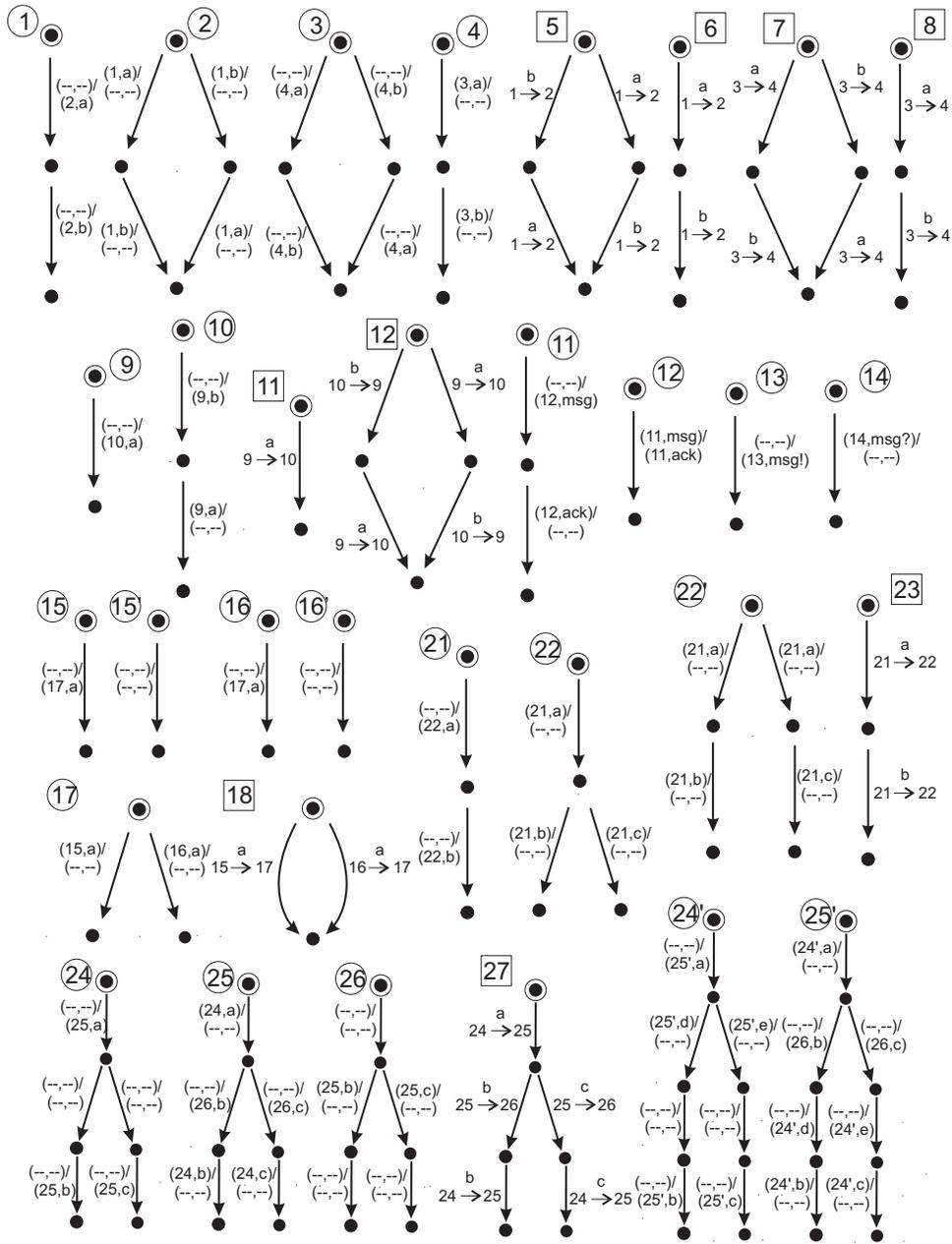


Figure 4. Orchestrations and choreographies.

Appendix A: Examples and Case Study

In this appendix we illustrate the use of the conformance relations given in Definition 4.1 with some simple examples, and next we show a small case study including a more elaborated system. Intuitively, a complete path-closure (see Definition 3.1) is a set consisting of a (maximal) sequence as well as all of its prefixes. Let us note that the longest element of a *finite* complete path-closure of traces necessarily finishes with the `stop` symbol. For the sake of clarity, from now on a complete path-closure will be referred just by its longest element not including the `stop` symbol. For instance, the complete path-closure $\{[], [(1, a, 2)], [(1, a, 2), (1, b, 2)], [(1, a, 2), (1, b, 2), \text{stop}]\}$ will be referred just by $[(1, a, 2), (1, b, 2)]$ (and we will say that $[(1, a, 2), (1, b, 2)]$ is a *complete trace*). Following a similar idea, an *infinite* complete path-closure of the form $\{[], [(a_1, b_1, c_1)], [(a_1, b_1, c_1), (a_2, b_2, c_2)], [(a_1, b_1, c_1), (a_2, b_2, c_2), (a_3, b_3, c_3)], \dots\}$ will be referred by the infinite list $[(a_1, b_1, c_1), (a_2, b_2, c_2), (a_3, b_3, c_3), \dots]$.

Figure 4 presents several orchestration services and choreographies. For all depicted services we will assume each input belongs to a different type of inputs. Let S_1 be a system of orchestration services consisting of services 1 and 2. We check whether S_1 conforms to choreographies 5 and 6. If we consider the conf_s relation, then we observe that S_1 conforms to both 5 and 6. This is because the only possible complete sending trace of S_1 is $[(1, a, 2), (1, b, 2)]$, which is included in the set of complete traces of 5 (which is $\{[(1, a, 2), (1, b, 2)], [(1, b, 2), (1, a, 2)]\}$) and 6 ($\{[(1, a, 2), (1, b, 2)]\}$). Concerning *full* conformance, we have that S_1 fully conforms to 6 with respect to sending traces, but not to 5. Regarding processing traces, let us note that S_1 can generate the complete processing traces $[(1, a, 2), (1, b, 2)]$ and $[(1, b, 2), (1, a, 2)]$ (note that, after a and b are received in the input buffer of service 2, service 2 can process them in any order). Both complete processing traces are included in the set of complete traces of 5, but not in the corresponding set of 6, which only includes $[(1, a, 2), (1, b, 2)]$. Thus, if either conf_p or conf_p^f are considered, then S_1 conforms to 5, but not to 6.

Let S_2 be the system consisting of services 3 and 4, and let us compare it with choreographies 7 and 8. In this case, we have the opposite result as before. In particular, if processing traces are considered, then S_2 conforms to both choreographies (if *full* conformance is considered, it only conforms to 8). However, S_2 does not conform to 8 when sending traces are considered, regardless of whether full conformance is considered or not. Let us note that S_2 can perform the sending traces $[(3, a, 4), (3, b, 4)]$ and $[(3, b, 4), (3, a, 4)]$. However, the sets of complete traces of choreographies 7 and 8 are $\{[(3, a, 4), (3, b, 4)], [(3, b, 4), (3, a, 4)]\}$ and $\{[(3, a, 4), (3, b, 4)]\}$, respectively. Thus, if conf_s or conf_s^f are considered, then S_2 conforms to choreography 7, but not to choreography 8.

Next, let S_3 be the system consisting of services 9 and 10. We compare S_3 with choreographies 11 and

12. The set of complete sending traces of S_3 is equal to $\{[(9, a, 10), (9, b, 10)], [(9, b, 10), (9, a, 10)]\}$, while the set of complete processing traces of S_3 is $\{[(9, a, 10)]\}$. On the one hand, the only complete trace of choreography 11 is $[(9, a, 10)]$, so S_3 conforms to 11 only if processing traces are considered (with respect to both conf_p and conf_p^f). On the other hand, choreography 12 can produce both $[(9, a, 10), (9, b, 10)]$ and $[(9, b, 10), (9, a, 10)]$. Since only complete traces are considered, S_3 conforms to 12 only if sending traces are regarded (according to both conf_s and conf_s^f).

Despite of the fact that only asynchronous communications are considered in our framework, synchronous communications can be trivially defined indeed. Let us consider the system S_4 consisting of services 13 and 14. After 13 sends message msg to 14, service 13 will be blocked until 14 performs its unique transition and sends message ack back to 13. So, a synchronous communication between 13 and 14 is actually expressed by this trivial structure. A syntactic sugar to denote a synchronous communication like this is implicitly proposed in pictures of services 15 and 16, which are intended to be equivalent to 13 and 14, respectively. In particular, we propose to denote a synchronous communication on message msg by using new symbols $msg?$ and $msg!$. Let us note that if only these kind of messages are used in orchestrations, then $\text{conf}_s = \text{conf}_p = \text{conf}$ and $\text{conf}_s^f = \text{conf}_p^f = \text{conf}^f$.

Let us recall that the suitability of an orchestration service to fulfill a given choreography depends on the behavior of the rest of involved services. In order to illustrate this, we revisit the travel agency example presented in Section 2. A travel agency (service 17) waits for a message a (standing for “we provide you with a transfer service”) from any of two possible services: an air company or a hotel. We consider two possible air companies, represented by services 15 and 15'. Service 15 provides service 17 with a transfer service, while 15' does nothing. Similarly, services 16 and 16' represent two hotels, where only 16 provides the travel agency with a transfer. Most combinations of, on one hand, either 15 or 15' and, on the other hand, either 16 or 16', allow 17 to satisfy the choreography 18 with respect to (non-full) sending and processing conformance. In fact, only combining 15' with 16' fails to meet both non-full semantic relations. Thus, either the air company or the hotel must provide the transfer. If full conformance is required, then the only valid combination of air company and hotel consists in taking 15 and 16, respectively.

We show that systems of orchestrations are required to *complete* all started sequences, that is, they are required not to finish a started sequence until the choreography explicitly allows it. Let us consider orchestration services 21, 22, and 22', as well as choreography 23. Let S_5 be a system consisting of services 21 and 22. The sequence $[(21, a, 22), (21, b, 22)]$ is both the only complete sending trace and the only complete processing trace of S_5 . Thus, S_5 conforms to choreography 23 with respect to both kinds

of traces. Let us substitute the definition of service 22 by that given for service 22', and let S'_5 be the resulting system. The set of complete sending traces of S'_5 is the same as S_5 , so S'_5 also conforms to 23 with respect to sending traces. However, the set of complete processing traces of S'_5 is $\{[(21, a, 22'), (21, b, 22')], [(21, a, 22')]\}$ because 22' could take its right path and get stuck after receiving a (more formally, $[(21, a, 22'), \text{stop}]$ is a processing trace of S'_5). Since $[(21, a, 22')]$ is not a *complete* processing trace of 23, S'_5 does not conform to 23 with respect to processing traces.

Finally, we consider a case where there are *infinite* complete traces in systems due to the presence of loops. Let us revisit the orchestrations and the choreography previously depicted in Figure 1, and let S be the composition of A and B . The infinite set of complete traces of choreography C is $T = \{\sigma, \sigma_1, \sigma_2, \sigma_3, \dots\}$, where σ is the infinite concatenation of the subsequence $\alpha = [(A, \text{request}, B), (B, \text{response}, A)]$, that is, $\sigma = \alpha \cdot \alpha \cdot \alpha \cdot \dots$, and for all $i \in \mathbf{N}$ we have $\sigma_i = (\alpha)^i \cdot (A, \text{exit}, B)$. In fact, the infinite set of complete sending and processing traces of S is T as well, so we have that S conforms to C with respect to all relations conf_s , conf_p , conf , conf_s^f , conf_p^f , and conf^f .

Finally, we present a small case study consisting in a more elaborated system. This 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 carrier. The purchase works as follows: “A customer wants to buy a product by using Internet. There are several sellers that offer different products in Internet Servers based on Web-pages. The customer contacts a seller in order to buy the desired product. The seller checks the stock and contacts with a carrier. Finally, the carrier delivers the product to the customer.”

Figures 5 and 6 depict the orchestration of the three actors represented in this purchase process, that is, the customer, the seller and the carrier. 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, it waits to receive the product from a carrier.
- 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 with a carrier to deliver the product.
- Carrier: It picks up the order and the customer information in order to deliver the product to the customer.

Figure 7 shows the choreography of this Internet purchase process. Once the orchestrations of the three service and the choreography specification are defined, we use the conformance relations given in Definition 4.1 to check if the com-

s_1	(1,iProduct,2), (2,lProduct,1), (1,Nothing,2)
s_2	(1,iProduct,2), (2,lProduct,1), (1,bProduct), (2,NoStock,1)
s_3	(1,iProduct,2), (2,lProduct,1), (1,bProduct), (2,Stock,1), (1,iPayment,2), (2,Receipt,1), (2,PickOrder,3), (3,DeliverOrder,1)

Table 1. The choreography C traces.

position of the proposed orchestration services satisfies the choreography.

Let us consider the system $S = (1, 2, 3)$, where 1, 2, and 3 represent the client service, the seller service, and the carrier service, respectively. Let C be the choreography machine depicted in Figure 7, and let s_1, s_2, s_3 be the sequences depicted in Table 1. For all complete trace σ of C , σ is an infinite concatenation of subsequences $\sigma = \alpha_1 \cdot \alpha_2 \cdot \alpha_3 \cdot \alpha_4 \cdot \dots$ where for all $i \in \mathbf{N}$ we have $\alpha_i \in \{s_1, s_2, s_3\}$. Let us note that any complete sending or processing trace σ' of S must also be an infinite concatenation of subsequences s_1, s_2, s_3 . Hence, for all $\sigma' \in \text{Comp}(\text{sndTraces}(S)) \cup \text{Comp}(\text{prcTraces}(S))$ we have $\sigma' \in \text{Comp}(\text{traces}(C))$, and thus we have both $S\text{conf}_s C$ and $S\text{conf}_p C$, which implies $S\text{conf} C$. Moreover, in this case we also have that, for all $\sigma \in \text{Comp}(\text{traces}(C))$, $\sigma \in \text{Comp}(\text{sndTraces}(S))$ and $\sigma \in \text{Comp}(\text{prcTraces}(S))$. Therefore, we also have $S\text{conf}_s^f C$, $S\text{conf}_p^f C$, and $S\text{conf}^f C$.

Appendix B: Derivation of decentralized systems of services

In this appendix we formally present the derivation of decentralized systems of services from choreographies. Two alternatives are considered: Making the system conform to the choreography with respect to all proposed conf_x^f conformance relations, and making it conform only with respect to sending traces. Theorems 5.2 and 5.4 show the correctness of both approaches.

Definition 5.1 Let $C = (S, M, ID, s_{in}, T)$ be a choreography machine where $ID = \{id_1, \dots, id_n\}$ and $S = \{s_1, \dots, s_l\}$. For all $s \in S$ and $id \in ID$, let $T_{s,id} = \{(s, m, id, adr, s') \mid \exists adr, m, s' : (s, m, id, adr, s') \in T\}$ and $m_{s,id} = |T_{s,id}|$. For all $1 \leq j \leq m_{s,id}$, let $t_{s,id,j}$ denote the j -th transition of $T_{s,id}$ according to some arbitrary ordering criterium. Let $[id_{a_1}^s, \dots, id_{a_{h_s}}^s]$ denote the sequence of all identifiers $id \in ID$ such that $m_{s,id} \geq 1$, ordered according to some arbitrary ordering criterium.

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

[CASE 1] If there are transitions leaving s in which id_i is the sender, but id_i is neither the first nor the last service

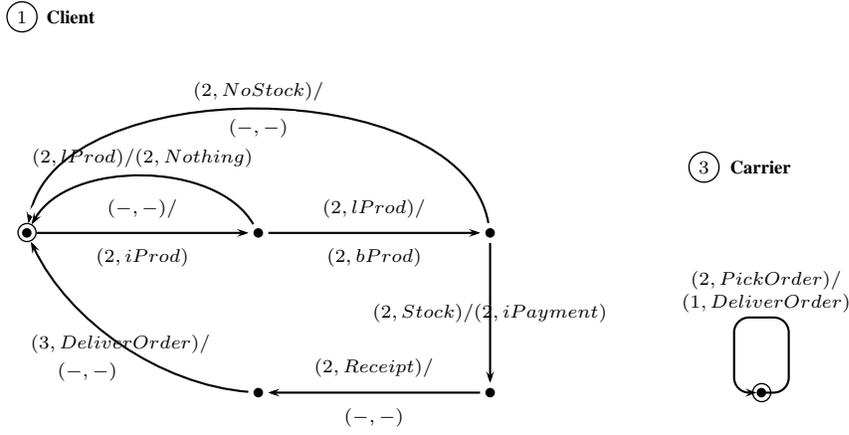


Figure 5. Client and Carrier orchestration specifications.

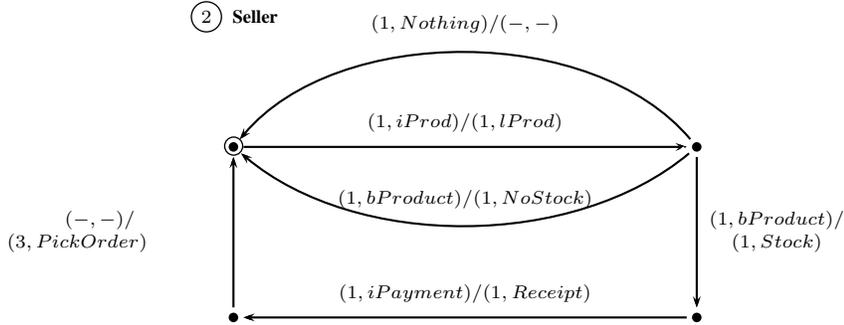


Figure 6. Seller orchestration specification.

doing so, that is, if $id_i = id_{a_{s_j}}^s$ for some $1 < j < a_{h_s}^s$, then we consider the following transitions:

- (a) $s \xrightarrow{(id_{i-1}, idontchoose)/(null, null)} s_{icanchoose}$ (id_{i-1} tells id_i that it refuses to choose one of its transitions).
- (b) $s \xrightarrow{(id_{i-1}, alreadychosen)/(id_{i+1}, alreadychosen)} s_{idontchoose}$ (id_{i-1} tells id_i that somebody has already chosen, and id_i propagates the message).
- (c) $s_{icanchoose} \xrightarrow{(null, null)/(id_{i+1}, idontchoose)} s_{idontchoose}$ (id_i decides not to choose).
- (d) $s_{icanchoose} \xrightarrow{(null, null)/(id_{i+1}, alreadychosen)} s_{iwillchoose}$ (id_i decides to choose).
- (e) For all $1 \leq j \leq T_{s, id}$, we have:
 - (e.1) Let $t_{s, id, j} = (s, m, snd, adr, s')$. We have $s_{iwillchoose} \xrightarrow{(null, null)/(adr, takemychoice_j)} s'$
 - (e.2) $s_{ichoose_j} \xrightarrow{(null, null)/(adr, m)} s_{ichoose'_j}$ (id_i sends the message m denoted by its j -th transition to adr).
 - (e.3) $s_{ichoose'_j} \xrightarrow{(adr, ididit)/(null, null)} s_{ichoose_{j1}}$ (id_i waits for a signal from adr indicating that m was processed).
 - (e.4) Let $G = \{g | g \in [1..n-1], g \neq i, id_g \neq adr\}$. For all $1 \leq j \leq T_{s, id}$ and all $k \in G$ we have
 - * $s_{ichoose_{jk}} \xrightarrow{(null, null)/(id_k, takemychoice_j)} s_{ichoose_{j k'}}$, where k' is the minimum value in G such that $k' > k$ (id_i asks everybody to take its choice).
 - (e.5) $s_{ichoose_{jn}} \xrightarrow{(null, null)/(id_n, takemychoice_j)} s'$ (after asking id_n to take its choice, id_i reaches the

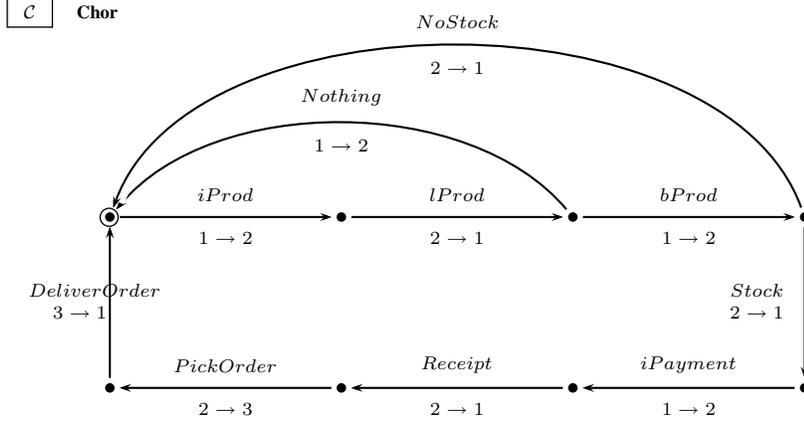


Figure 7. The choreography specification.

destination of $t_{s, id_i, j}$, that is s').

- (f) For all $j \in [1..n] \setminus \{i\}$ and for all $1 \leq k \leq T_{s, id_j}$, we have:

- (f.1) Let us assume $t_{s, id_j, k} = (s, m, snd, adr, s')$. If $adr = id_i$ then we have $s_{idontchoose} \xrightarrow{(id_j, takemychoice_k)/(null, null)}$ $s_{ifollow_{jk}}$ and $s_{ifollow_{jk}} \xrightarrow{(id_j, m)/(id_j, ididit)}$ s' (id_i takes the k -th choice of id_j , which makes id_i receive a message from id_j and next acknowledge it).

- (f.2) Otherwise, we have $s_{idontchoose} \xrightarrow{(id_j, takemychoice_k)/(null, null)}$ s' (id_i takes the k -th choice of id_j , which does not concern id_i).

[CASE 2] If there are transitions leaving s in which id_i is the sender and id_i is the first service doing so, that is, if $id_i = id_{a_{s1}}^s$, then we consider the same transitions as in case 1, though transitions given in (a) and (b) are replaced by $s \xrightarrow{(null, null)/(null, null)}$ $s_{icanchoose}$.

[CASE 3] If there are transitions leaving s in which id_i is the sender and id_i is the last service doing so, that is, if $id_i = id_{a_{hs}^s}$, then we consider the same transitions as in case 1, though the transition given in (b) is replaced by $s \xrightarrow{(id_{i-1}, alreadychosen)/(null, null)}$ $s_{idontchoose}$, the transition denoted in (c) is removed, and transition (d) is replaced by $s_{icanchoose} \xrightarrow{(null, null)/(null, null)}$ $s_{iwillchoose}$.

[CASE 4] If there is no transition leaving s in which id_i is the sender, that is if $id_i \neq id_{a_{sj}^s}$ for all $1 \leq j \leq a_{hs}^s$, then we consider the same transitions as in case 1, though transitions given in (a) and (b) are replaced by $s \xrightarrow{(null, null)/(null, null)}$ $s_{idontchoose}$. \square

Theorem 5.2 Let $\mathcal{C} = (S, M, ID, s_{in}, T)$ be a choreography with $ID = \{id_1, \dots, id_n\}$. Let $\mathcal{S} = (\text{decentral}(\mathcal{C}, id_1), \dots, \text{decentral}(\mathcal{C}, id_n))$. For all $\text{conf}_x \in \{\text{conf}'_s, \text{conf}'_p, \text{conf}', \text{conf}^{f'}_s, \text{conf}^{f'}_p, \text{conf}^{f'}\}$ we have $\mathcal{S} \text{ conf}_x \mathcal{C}$. \square

Definition 5.3 We have that $\text{decentral}'(\mathcal{C}, id_i)$ is defined as $\text{decentral}(\mathcal{C}, id_i)$ in Definition 5.1 after replacing the transition (e.3) by $s_{ichoose'_j} \xrightarrow{(null, null)/(null, null)}$ $s_{ichoose_{j1}}$ and replacing the second transition denoted in (f.1) by $s_{ifollow_{jk}} \xrightarrow{(id_j, m)/(null, null)}$ s' . \square

Theorem 5.4 Let $\mathcal{C} = (S, M, ID, s_{in}, T)$ be a choreography with $ID = \{id_1, \dots, id_n\}$. Let $\mathcal{S} = (\text{decentral}'(\mathcal{C}, id_1), \dots, \text{decentral}'(\mathcal{C}, id_n))$. For all $\text{conf}_x \in \{\text{conf}'_s, \text{conf}^{f'}_s\}$ we have $\mathcal{S} \text{ conf}_x \mathcal{C}$. \square