Realizability is controllability

Niels Lohmann and Karsten Wolf

Universität Rostock, Institut für Informatik, 18051 Rostock, Germany
{niels.lohmann, karsten.wolf}@uni-rostock.de

Abstract. A *choreography* describes the interaction between services. It may be used for specification purposes, for instance serving as a contract in the design of an interorganizational business process. Typically, not all describable interactions make sense which motivates the study of the *realizability* problem for a given choreography.
In this paper, we show that realizability can be traced back to the problem of *controllability* which askes whether a service has compatible partner processes. This way of thinking makes algorithms for controllability available for reasoning about realizability. In addition, it suggests alternative definitions for realizability. We discuss several proposals for defining realizability which differ in the degree of coverage of the specified interaction.

1 Introduction

Dumas et al. discuss in [1] compatibility between services and introduce a number of related notions. Among these notions they mention *realizability* (the problem whether a choreography can be implemented by services) and *controllability* (the problem whether a service has a compatible partner), but state that "... a formal relation between controllability and realizability is yet to be established". This paper is dedicated to the establishment of this relation.

In Sect. 2, we introduce a formal framework which allows us to reason about choreographies in a formal and language-independent manner. In Sect. 3, we recall different realizability notions and introduce the novel concept of *distributed realizability*, which seamlessly complements existing notions. Section 4 formulates the realizability problem in terms of controllability and shows how algorithms for controllability can be used to proof realizability by synthesizing realizing services. Section 5 discusses further research questions and concludes the paper.

2 A Formal Framework for Choreographies

Throughout this paper, fix a finite set of message channels M that is partitioned into asynchronous message channels M_A and synchronous message channels M_S. From M, derive a set of message events $E := !E \cup ?E \cup !?E$, consisting of asynchronous send events $!E := \{!x \mid x \in M_A\}$, asynchronous receive events $?E := \{?x \mid x \in M_A\}$, and synchronization events $!?E := \{!?x \mid x \in M_S\}$.

Definition 1 (Conversation, choreography). *A* conversation γ *is a finite word over* E *such that, for all* $x \in M_A$, $\#_{!x}(\gamma) = \#_{?x}(\gamma)$ *and for every prefix* γ' *of* γ *holds:* $\#_{!x}(\gamma') \geq \#_{?x}(\gamma')$. *Thereby,* $\#_x(\gamma)$ *denotes the number of occurrences of the message event* x *in the word* γ. *A* choreography *is a set of conversations.*

The requirements for a conversation state that asynchronous events are always paired, and a send event always occurs before the respective receive event. A choreography is a set of desired message event sequences. A choreography is defined with respect to a set of peers which form a collaboration.

Definition 2 (Peer, collaboration). *A* peer $P = [I, O]$ *consists of a set of input message channels* $I \subseteq M$ *and a set of output message channels* $O \subseteq M$, $I \cap O = \emptyset$. *A* collaboration *is a set* $\{P_1, \ldots, P_n\}$ *of peers such that* $I_i \cap I_j = \emptyset$ *and* $O_i \cap O_j = \emptyset$ *for all* $i \neq j$, *and* $\bigcup_{i=1}^{n} I_i = \bigcup_{i=1}^{n} O_i$.

The requirements ensure that communication in a collaboration is always bilateral, yielding a closed system that models no message exchange other than that between the peers.

There are various languages to specify collaborations and choreographies, ranging from formal models such as *Interaction Petri Nets* [2] to industrial notations such as *Let's Dance* [3], UML collaboration diagrams, and BPMN. While these languages differ in syntax and semantics, concepts such as an underlying collaboration (i.e., the endpoints and the exchanged messages) and the choreography (i.e., the intended global behavior) can be easily derived from these languages.

To specify the behavior of a service (i.e., the concrete implementation of a peer), again a variety of languages exist. In essence, they all share a concept of a state (e.g., a state of a finite state machine, a marking of a Petri net, or control flow tokens in BPMN) and ways to specify message transfer. These basic concepts can be expressed with service automata.

Definition 3 (Service automaton). *A* service automaton $A = [Q, I, O, \delta, q_0, F]$ *is a tuple such that* Q *is a finite set of states,* $[I, O]$ *is a peer,* $\delta \subseteq Q \times (E_I \cup E_O \cup \{\tau\}) \times Q$ *is a transition relation,* $q_0 \in Q$ *is an initial state, and* $F \subseteq Q$ *is a set of final states. Thereby,* $E_I := \{?x \mid x \in I \cap M_A\} \cup \{!?x \mid x \in I \cap M_S\}$ *and* $E_O := \{!x \mid x \in O \cap M_A\} \cup \{!?x \mid x \in O \cap M_S\}$.

We say that A implements $[I, O]$, and for $(q, x, q') \in \delta$, we also write $q \xrightarrow{x} q'$. Beside internal (i.e., silent, non-communicating) τ steps and synchronous communication, service automata also model asynchronous communication, in which messages may overtake each other, but will never get lost. We claim that is — compared to FIFO queues for communicating finite state machines [4] — a more natural approach to model asynchronicity, because it makes less assumptions about the underlying infrastructure. In the composition of two or more service automata, pending asynchronous messages are represented by a multiset. Denote the set of all multisets over M_A with $Bags(M_A)$. Further denote the empty multiset with $[\,]$, and the multiset containing only one instance of $x \in M_A$ with $[x]$. Addition of multisets is defined pointwise.

Definition 4 (Composition of service automata). *Let $A_1, \ldots, A_n$ be service automata such that their peers form a collaboration. Define the* composition *$A_1 \oplus \cdots \oplus A_n$ as the automaton $[Q, \delta, q_0, F]$ with $Q := Q_1 \times \cdots \times Q_n \times Bags(M_A)$, $q_0 := [q_{0_1}, \ldots, q_{0_n}, [\,]]$, $F := F_1 \times \cdots \times F_n \times \{[\,]\}$, and, for all $i \neq j$ the transition relation δ contains excactly the following elements:*

- $[q_1, \ldots, q_i, \ldots, q_n, B] \xrightarrow{\tau} [q_1, \ldots, q_i', \ldots, q_n, B]$, *iff* $[q_i, \tau, q_i'] \in \delta_i$ *(internal move by A_i),*
- $[q_1, \ldots, q_i, \ldots, q_n, B] \xrightarrow{!x} [q_1, \ldots, q_i', \ldots, q_n, B + [x]]$, *iff* $[q_i, !x, q_i'] \in \delta_i$ *(asynchronous send by A_i),*
- $[q_1, \ldots, q_i, \ldots, q_n, B + [x]] \xrightarrow{?x} [q_1, \ldots, q_i', \ldots, q_n, B]$, *iff* $[q_i, ?x, q_i'] \in \delta_i$ *(asynchronous receive by A_i), and*
- $[q_1, \ldots, q_i, \ldots, q_j, \ldots, q_n, B] \xrightarrow{!?x} [q_1, \ldots, q_i', \ldots, q_j', \ldots, q_n, B]$, *iff* $[q_i, !?x, q_i'] \in \delta_i$ *and* $[q_j, !?x, q_j'] \in \delta_j$ *(synchronization between A_i and A_j).*

A run of $A_1 \oplus \cdots \oplus A_n$ is a sequence of events $x_1 \cdots x_m$ such that $q_0 \xrightarrow{x_1} \cdots \xrightarrow{x_m} q_f$ with $q_f \in F$. For a run ρ, define the conversation of ρ as $\rho_{|E}$. The choreography of $A_1 \oplus \cdots \oplus A_n$, denoted $Chor(A_1 \oplus \cdots \oplus A_n)$, is the union of the conversations of all runs of $A_1 \oplus \cdots \oplus A_n$.

The composition is finite iff, for each state, the number of pending asynchronous messages is bounded. The choreography of a composition of service automata is the set of all observable event sequences that are produced by runs of the composition.

3 Realizability Notions

With the notion of realizability, the conversations generated by a composition of services can be related to a specified choreography. Bultan et al. [5, 6] define realizability of choreographies as follows:

Definition 5 (Complete realizability). *A choreography C is* completely realizable *w.r.t. a collaboration $\{P_1, \ldots, P_n\}$ iff there exists a tuple of service automata $[A_1, \ldots, A_n]$ such that, for all i, A_i implements P_i, and $Chor(A_1 \oplus \cdots \oplus A_n) = C$.*

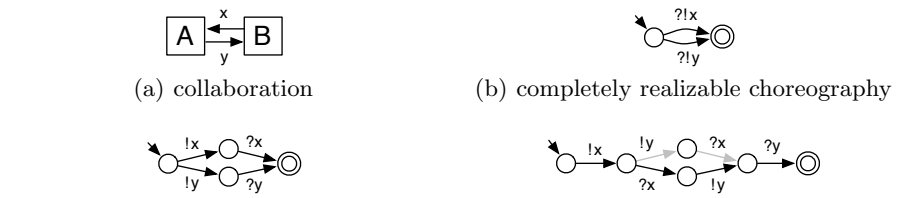
Complete realizability is a strong requirement, because it demands that the observable behavior of the endpoints exactly matches the choreography. In reality, it is often the case that not all aspects of a choreography can be implemented. To this end, Zaha et al. [7] introduce a weaker notion called *local enforceability* (or *partial realizability*) which only demands that a subset of the choreography is realized by the peer implementations:

Definition 6 (Partial realizability). *A choreography C is* partially realizable *w.r.t. a collaboration $\{P_1, \ldots, P_n\}$ iff there exists a tuple of service automata $[A_1, \ldots, A_n]$ such that, for all i, A_i implements P_i, and $Chor(A_1 \oplus \cdots \oplus A_n) \subseteq C$.*

Obviously, complete realizability implies partial realizability. Though this weaker notion ensures that all constraints of the choreography are fulfilled, it still fixes a single tuple of service automata. If there does not exist such tuple of automata that realizes the *complete* choreography, there might still exist a *set* of tuples — each partially realizing the choreography — which distributedly realizes the complete choreography:

Definition 7 (Distributed realizability). *A choreography C is* distributedly realizable *w.r.t. a collaboration $\{P_1, \ldots, P_n\}$ iff there exist tuples of service automata $[A_{1_1}, \ldots, A_{n_1}], \ldots, [A_{1_m}, \ldots, A_{n_m}]$ such that, for $i = 1, \ldots, n$ and $j = 1, \ldots, m$, (i) A_{i_j} implements P_i, (ii) $Chor(A_{1_j} \oplus \cdots \oplus A_{n_j}) \subseteq C$, and (iii) $\bigcup_{j=1}^{m} Chor(A_{1_j} \oplus \cdots \oplus A_{n_j}) = C$.*

Distributed realizability allows for build time coordination between peers. While being a stronger notion than partial realizability (i.e., more of choreography's behavior is implemented), it is still a weaker notion than complete realizability.



(a) collaboration

(b) completely realizable choreography

(c) distributedly realizable choreography

(d) partially realizable choreography

Fig. 1. Example choreographies with respect to a collaboration.

As an example, consider the collaboration depicted in Fig. 1(a) in which two peers, A and B, communicate via message channels x and y. The choreography {!?x, !?y} in which the peers communicate synchronously (b) is completely realizable by a set of peers which synchronously decided whether to synchronize via message x or y. In case the messages are sent asynchronously (c), this is no longer possible. This choreography is not completely realizable, because there does not exist a single pair of service automata that implement the specified behavior. Instead, the implementations have to be coordinated: either peer A sends a message and peer B is quiet or the other way around. These two pairs distributedly realize the whole choreography. Finally, choreography (d) can only be partially realized, because the conversation !x!y?x?y cannot be implemented even if the peers are coordinated at build time. This is because the requirement that message x is sent before message y cannot be enforced, because the peers cannot coordinate this.

4 Synthesizing Realizing Peer Implementations

In this section, we show how the different realizability notions are related to controllability [8]. Controllability is a correctness criterion for services: a service A is controllable iff there exists a service B such that $A \oplus B$ is compatible (i.e.,

deadlock free). Controllability can be extended to multi-port services. In the following, we will transform a choreography into a multi-port service which is controllable if and only if the choreography is realizable.

For a regular[1] choreography C, there exists a deterministic finite state machine that accepts exactly the sequences of the choreography. We call this state machine the *monitor* for C [9]. It unobtrusively monitors the interaction between the peers and reaches a final state iff the monitored conversation was part of the choreography.

Definition 8 (Monitor, monitored composition). *Let C be a regular choreography w.r.t. a collaboration $\{P_1, \ldots, P_n\}$. Define the deterministic finite state machine accepting C (the* monitor *for C) as $M = [Q_M, \delta_M, q_{0_M}, F_M]$. Thereby, Q_M is a finite set of states, $\delta_M : Q_M \times E \to Q_M$ is a transition function, $q_{0_M} \in Q_M$ is an initial state, and $F_M \subseteq Q_M$ is a set of final states.*

Let $A_1, \ldots, A_n$ be service automata implementing $P_1, \ldots, P_n$. Define the monitored composition *$M \otimes (A_1 \oplus \cdots \oplus A_n)$ as the automaton $[Q, \delta, q_0, F]$ with $Q := Q_M \times Q_1 \times \cdots \times Q_n \times Bags(M_A)$, $q_0 := [q_{0_M}, q_{0_1}, \ldots, q_{0_n}, [\,]]$, $F := F_M \times F_1 \times \cdots \times F_n \times [\,]]$, and, for all $i \neq j$, the transition relation δ contains excactly the following elements:*

- $[q, q_1, \ldots, q_i, \ldots, q_n, B] \xrightarrow{\tau} [q, q_1, \ldots, q_i', \ldots, q_n, B]$, *iff* $[q_i, \tau, q_i'] \in \delta_i$ *(internal move by A_i),*
- $[q, q_1, \ldots, q_i, \ldots, q_n, B] \xrightarrow{!x} [q', q_1, \ldots, q_i', \ldots, q_n, B + [x]]$, *iff* $[q_i, !x, q_i'] \in \delta_i$ *and* $[q, !x, q'] \in \delta_M$ *(asynchronous send by A_i, monitored by M),*
- $[q, q_1, \ldots, q_i, \ldots, q_n, B + [x]] \xrightarrow{?x} [q', q_1, \ldots, q_i', \ldots, q_n, B]$, *iff* $[q_i, ?x, q_i'] \in \delta_i$ *and* $[q, !?, q'] \in \delta_M$ *(asynchronous receive by A_i, monitored by M),*
- $[q, q_1, \ldots, q_i, \ldots, q_j, \ldots, q_n, B] \xrightarrow{!?x} [q', q_1, \ldots, q_i', \ldots, q_j', \ldots, q_n, B]$, *iff* $[q_i, !?x, q_i'] \in \delta_i$, $[q_j, !?x, q_j'] \in \delta_j$, *and* $[q, !?x, q'] \in \delta_M$ *(synchronization between A_i and A_j, monitored by M).*

The monitor synchronizes with the message events of the service automata, but does not constrain their behavior. The monitor only has an effect on the final states of the composition. Only if all service automata *and* the monitor reach a final state, this state is final in the monitored composition.

We can now change the point of view and regard the monitor as a service that is communicating with several other services by synchronous message events. Again, this service will reach a final state iff the message events from the environment are observed in the correct order.

Definition 9 (Monitor service automaton). *Let C be a regular choreography w.r.t. a collaboration $\{P_1, \ldots, P_n\}$ and let $M = [Q_M, \delta_M, q_{0_M}, F_M]$ be a monitor for C. Define the* monitor service automaton *$A_M := [Q, \mathcal{I}, \mathcal{O}, \delta, q_0, F]$ with $Q := Q_M$, $\mathcal{I} := \{I_1, \ldots, I_n\}$, $\mathcal{O} := \{O_1, \ldots, O_n\}$, $q_0 := q_{0_M}$, and $F := F_M$. Define $\delta : Q \times \{!?\langle x\rangle \mid x \in E\} \to Q$ with $\delta(q, !?\langle x\rangle) := \delta_M(q, x)$.*

[1] Choreographies specified by interaction Petri nets, UML collaboration diagrams, BPMN, or Let's Dance are always regular if they assume synchronous communication.

Due to the nature of a choreography to exist of message events (not the messages itself), all message events of the monitor service automaton are synchronous. The original nature of the event (synchronous or asynchronous) is, however, encoded in the events by using the event $?!\langle x \rangle$ for event x. The existence of service automata that are compatible to this monitor service automaton proof realizability of the choreography:

Theorem 1. *Let C be a regular choreography w.r.t. a collaboration $\{P_1, \ldots, P_n\}$ and A_M a monitor service automaton for C.*

1. *C is partially realizable iff A_M is decentralized controllable.*
2. *C is distributedly realizable iff A_M is decentralized controllable and for the set of strategies $\mathcal{S}$ holds: $\bigcup_{[A_1,\ldots,A_n]\in\mathcal{S}} Chor(A_1 \oplus \cdots \oplus A_n) = C$.*
3. *C is completely realizable iff A_M is decentralized controllable and there exists a strategy $[A_1, \ldots, A_n]$ such that $Chor(A_1 \oplus \cdots \oplus A_n) = C$.*

A strategy for the monitor service automaton is a set of service automata such that the overall composition is compatible. While these automata communicate solely synchronously with the monitor service automaton, a tuple of service automata that actually realize the original choreography can be derived by changing every message event $!?\langle x \rangle$ back to x (e.g. "$!?\langle !x \rangle$" to "$!x$"). These service automata then can also communicate asynchronously with other peers, yet still follow the specified choreography.

5 Conclusion

In this paper, we linked the realizability problem to controllability, making existing tools and algorithms applicable to check choreographies. In particular, our approach allows for specification and synthesis of asynchronous peer implementations. This avoids a subsequent analysis and correction when trying to "asynchronize" peer implementations [10]. The current results are independent of a concrete choreography or service description language, but can be easily adapted.

In future work, we plan to study how choreography and service design can be mixed. For example, the realizability of a choreography can be checked w.r.t. some already completely specified peers. Such an approach would nicely complement the participant synthesis introduced in [11] by using controllability for both interaction models and interconnection models.

Acknowledgements This work is funded by the DFG project "Operating Guidelines for Services" (WO 1466/8-1).

References

1. Dumas, M., Benatallah, B., Nezhad, H.R.M.: Web service protocols: Compatibility and adaptation. IEEE Data Eng. Bull. **31**(3) (2008) 40–44

2. Decker, G., Weske, M.: Local enforceability in interaction petri nets. In: BPM 2007. LNCS 4714, Springer (2007) 305–319
3. Zaha, J.M., Barros, A.P., Dumas, M., Hofstede, A.H.M.t.: Let's dance: A language for service behavior modeling. In: OTM 2006. LNCS 4275, Springer (2006) 145–162
4. Brand, D., Zafiropulo, P.: On communicating finite-state machines. J. ACM **30**(2) (1983) 323–342
5. Fu, X., Bultan, T., Su, J.: Conversation protocols: a formalism for specification and verification of reactive electronic services. Theor. Comput. Sci. **328**(1-2) (2004) 19–37
6. Bultan, T., Fu, X.: Specification of realizable service conversations using collaboration diagrams. SOCA **2**(1) (2008) 27–39
7. Zaha, J.M., Dumas, M., Hofstede, A.H.M.t., Barros, A.P., Decker, G.: Service interaction modeling: Bridging global and local views. In: EDOC 2006, IEEE Computer Society (2006) 45–55
8. Wolf, K.: Does my service have partners? LNCS ToPNoC **II**(5460) (2009) 152–171
9. Lohmann, N., Massuthe, P., Wolf, K.: Behavioral constraints for services. In: BPM 2007. LNCS 4714, Springer (2007) 271–287
10. Decker, G., Barros, A., Kraft, F.M., Lohmann, N.: Non-desynchronizable service choreographies. In: ICSOC 2008. LNCS 5364, Springer (2008) 331–346
11. Lohmann, N., Kopp, O., Leymann, F., Reisig, W.: Analyzing BPEL4Chor: Verification and participant synthesis. In: WS-FM 2007. LNCS 4937, Springer (2008) 46–60