Choreography Conformance via Synchronizability *

Samik Basu
Department of Computer Science
Iowa State University
Ames, IA 50011, USA
sbasu@cs.iastate.edu

Tevfik Bultan
Department of Computer Science
University of California
Santa Barbara, CA 93106, USA
bultan@cs.ucsb.edu

ABSTRACT

Choreography analysis has been a crucial problem in service oriented computing. Interactions among services involve message exchanges across organizational boundaries in a distributed computing environment, and in order to build such systems in a reliable manner, it is necessary to develop techniques for analyzing such interactions. Choreography conformance involves verifying that a set of services behave according to a given choreography specification that characterizes their interactions. Unfortunately this is an undecidable problem when services interact with asynchronous communication. In this paper we present techniques that identify if the interaction behavior for a set of services remain the same when asynchronous communication is replaced with synchronous communication. This is called the synchronizability problem and determining the synchronizability of a set of services has been an open problem for several years. We solve this problem in this paper. Our results can be used to identify synchronizable services for which choreography conformance can be checked efficiently. Our results on synchronizability are applicable to any software infrastructure that supports message-based interactions.

Categories and Subject Descriptors

H.5.3 [**Group and Organization Interfaces**]: Asynchronous Interaction; D.2.4 [**Software/Program Verification**]: Model Checking; D.2.4 [**Software/Program Verification**]: Formal Methods

General Terms

Theory, Verification

Keywords

Message-based Interaction, Synchronizability

1. INTRODUCTION

Software systems are becoming increasingly more concurrent and distributed. In fact, nowadays, many software systems consist of multiple components that execute concurrently, possibly on different machines. Moreover, new trends in computing, such as service-oriented architecture, cloud computing, multi-core hardware, all point to even more concurrency and distribution among the components of software systems in the future. At the same time, concurrent and distributed software systems are increasingly used in every aspect of society and in some cases provide safety critical services. Hence, it is very important to develop techniques that guarantee that these software systems behave as they are expected to behave.

A crucial problem in dependability of concurrent and distributed software systems is the coordination of different components that form the whole system. In order to complete a task, components of a software system have to coordinate their executions by interacting with each other. A fundamental question is, what should be the interaction mechanism given the trend for increased level of concurrency and distribution in computing? One emerging paradigm is message-based communication [17, 21, 16, 24, 2, 5, 19, 8], where components interact with each other by sending and receiving messages.

Specification and analysis of message-based interactions has been an important research area in service oriented computing in the last several years. Choreography languages enable specification of such interactions. A choreography specification corresponds to a global ordering of the message exchange events among the peers participating to a composite service, i.e., a choreography specification identifies the set of allowable message sequences for a composite web service. Choreography conformance problem is identifying if a set of given services adhere to a given choreography specification. In general this conformance problem is undecidable when asynchronous communication is used. This is because, systems where peers communicate asynchronously with unbounded FIFO message queues can simulate Turing Machines [3].

In this paper, we identify a class of systems where choreography conformance can be efficiently checked even in the presence of asynchronous communication. We achieve this by checking a condition called synchronizability. A set of services is synchronizable if and only if the ordering of message exchanges remain the same when asynchronous communication is replaced with synchronous communication. In this paper we give an algorithm for determining synchronizability of a set of services that communicate with asynchronous communication.

It is important to note that the choreography analysis problem is not isolated to the area of service-oriented computing. It is a fundamental problem that appears in any

*This work is supported in parts by NSF grants CNS-0716095 and CCF-0702758.

Copyright is held by the International World Wide Web Conference Committee (IW3C2). Distribution of these papers is limited to classroom use, and personal use by others.
WWW 2011, March 28–April 1, 2011, Hyderabad, India.
ACM 978-1-4503-0632-4/11/03.

area where message-based communication is used to coordinate interactions of multiple concurrent or distributed components. For example, recently, earlier results on choreography analysis have been applied to analysis of Singularity channel contracts [22]. Singularity is an experimental operating system developed by Microsoft Research in order to improve the dependability of software systems [14]. In the Singularity operating system all inter-process communication is done via messages sent through asynchronous communication channels. Each channel is governed by a channel contract [8]. A channel contract is basically a state machine that specifies the allowable ordering of messages between the client and the server. Hence, channel contracts serve the same purpose that choreography specifications serve in service oriented computing.

As another example, UBF(B) is a specification language for specification of communication contracts in distributed Erlang programs [1]. UBF(B) contracts are finite state machines, where transitions correspond to request response patterns. Given a state, a transition from that state identifies a request response sequence where after receiving a message, the process sends a response and changes its state to the destination state.

UBF(B) contracts, Singularity channel contracts, and web service choreography specifications are all mechanisms for specifying ordering of messages exchanged among a set of concurrent or distributed processes. Analysis and verification of message-based interactions is an essential problem for all these specification mechanisms and the results we present in this paper are directly applicable to all of them.

1.1 A Summary of Our Results

The core problem we study in this paper is the following: Given a set of peers (individual services) that interact via asynchronous messaging (i.e., messages are sent and received through unbounded FIFO message queues), does the interaction behavior change when asynchronous communication is replaced with synchronous communication? This is called the synchronizability problem [11]. In asynchronous communication with unbounded message queues, the send actions are never blocked. In contrast, in synchronous communication each send action must synchronize with a corresponding receive action in order to execute, otherwise it blocks (this is also called rendezvous style communication). Synchronizability problem investigates the equivalence of asynchronous and synchronous communication as far as the interaction behavior is concerned. Interaction behavior is defined as the global sequence of send actions.

The synchronizability problem has been open for several years, in the sense that it was not known if synchronizability is decidable or not. Note that finite state machines communicating with unbounded message queues can simulate Turing machines, hence, many verification problems about them is undecidable [3]. In this paper we give an algorithm for determining synchronizability of a set of finite state peers that communicate with unbounded message queues. Our main result is that synchronizability can be determined by comparing the behavior of the peers with synchronous communication and with bounded asynchronous communication where each message queue is restricted to a queue of size 1 (i.e., if there is already a message in the queue, then the send actions that try to send to that queue block). We show that if the interaction behavior of the peers are the same for the synchronous and the 1-bounded asynchronous communication then they are synchronizable (i.e., the interaction behavior is also the same for unbounded asynchronous communication). If the interaction behavior of the peers are not the same for the synchronous and the 1-bounded asynchronous communication, then we also know that it is not the same for unbounded asynchronous communication. Hence, comparing the behavior of the synchronous communication and the 1-bounded asynchronous communication is enough for determining synchronizability. This type of comparison can be done using existing finite state verification tools since both synchronous communication and 1-bounded asynchronous communication lead to finite state spaces.

Algorithm 1 Conformance via Synchronizability

```
1: procedure ConfChecking(C, P_1, P_2, ..., P_n)
        ▷ C: Choreography; P_i: i-th Peer Description;
2:     Construct I_0;
3:     Construct I_1;
       ▷ I_i: System with peers using i size message queues
4:     if I_0 is Equivalent to I_1 then    ▷ Synchronizable
5:         Modelcheck C against I_0 (e.g., use Spin);
6:         return result;
7:     else
8:         System is not synchronizable
9:     end if
10: end procedure
```

Once we determine that a set of peers are synchronizable, then we can easily check choreography conformance using existing finite state verification tools. Algorithm 1 outlines this strategy. Note that choreography conformance checking is undecidable in general since, as we mentioned earlier, finite state machines communicating with unbounded message queues can simulate Turing machines, and therefore it is not possible to verify them automatically. The synchronizability checking mechanism we present in this paper enables us to identify a class of peers where choreography conformance can be easily checked. Interestingly many choreography specifications lead to synchronizable interactions, so this class seems to be practically very useful.

1.2 Organization

Rest of the paper is organized as follows. Section 2 presents some motivating examples and discusses the salient aspects of our contribution. Section 3 introduces the formal description of peers and systems, and describes the choreography conformance problem. Section 4 discusses the formalisms necessary for proving the decidability of synchronizability problem. Section 5 presents the main theorems identifying the necessary and sufficient conditions for synchronizability. Section 6 compares our technique with the existing ones. Finally, Section 7 summarizes the contributions of our work followed by some future avenues of research.

2. AN OVERVIEW

Figure 1 shows three variations of systems containing requester and server peers [11]. Each transition is labeled with either a send action (prefixed with "!") or a receive action (prefixed with "?"). The start and the final states in the requester peers are t_1 and t_2, respectively; while the start and the final states in the server peers are s_1 and s_2, re-

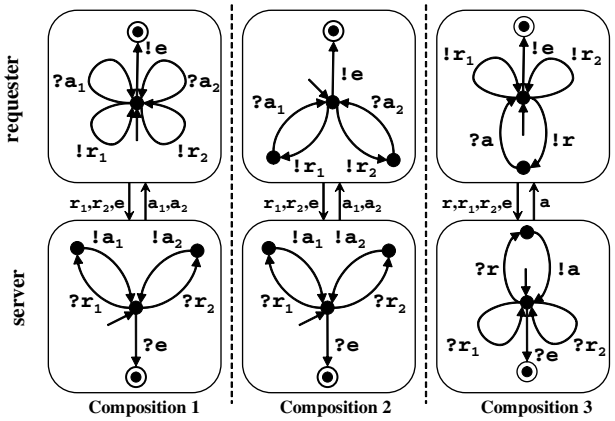


Figure 1: Three Systems with Server and Requester Peers [11]

spectively. In the first two systems, whenever the requester sends a request message (r_1 or r_2), the server responds with a corresponding acknowledgment (a_1 or a_2). Note that although the behavior of the server is identical in the first two systems the behavior of the requester is different. In the third system, one type of request message (r) causes server to send an acknowledgment message (a), but the other two types of requests (r_1 and r_2) are not acknowledged. Finally, when the requester sends an end message (e) the interaction terminates.

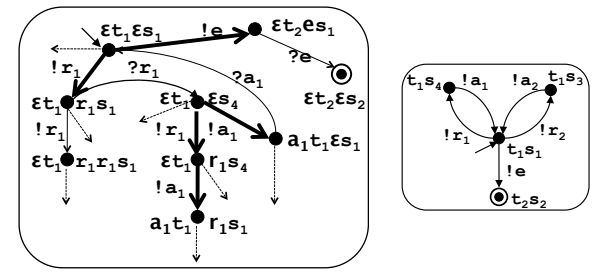
Figure 2 illustrates the behavior of each of the systems when the peers communicate asynchronously or synchronously. In asynchronous communication, when a peer executes a send action, the sent message is added to the tail of the receive queue of the receiver, and a peer can consume the message at the head of its receive queue by executing a receive action. The size of the queue is unbounded. In synchronous communication, on the other hand, sender and receiver move in lock-step, i.e., a send action of a sender is allowed only when the receiver is ready to perform the corresponding receive action. The message exchange occurs when the sender and the receiver take the send and receive transitions simultaneously.

In Figure 2, for the asynchronous case, we show the global state of the system by listing the local states of the participating peers along with the contents of their queues. We annotate each state by first listing the contents of the requester's receive queue (where ϵ means that the queue is empty), followed by the requester's local state, followed by the server's receive queue, followed by the server's local state. For instance, initially the queues of all peers are empty, and each peer is in its initial local state, i.e., the global system state is $(\epsilon t_1 \epsilon s_1)$.

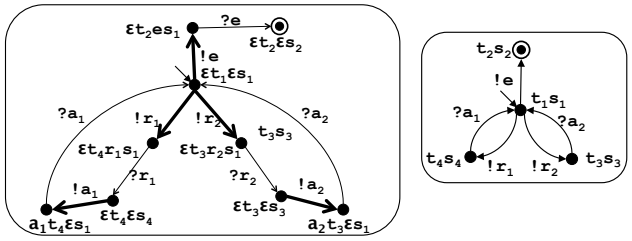
Observe that, asynchronous communication may result in a system which contains infinite number of states. For instance, when peers asynchronously communicate, the behavior of System 1 contains an infinite length sequence of states

$$\epsilon t_1 \epsilon s_1 \xrightarrow{!r_1} \epsilon t_1 r_1 s_2 \xrightarrow{!r_1} \epsilon t_1 r_1 r_1 s_2 \xrightarrow{!r_1} \dots$$

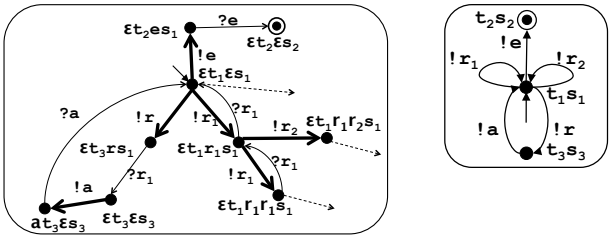
where requester peer keeps sending r_1 which is not consumed by the server. System 3 also exhibits similar infinite state behavior when peers communicate asynchronously (Figure 2(c)-i). As a result, it is not possible to directly verify whether Systems 1 and 3 (when peers communicate



(a) System 1: i. Asynchronous (partial) & ii. Synchronous



(b) System 2: i. Asynchronous & ii. Synchronous



(c) System 3: i. Asynchronous (partial) & ii. Synchronous

Figure 2: System Behaviors using Peers in Figure 1

asynchronously) conform to any choreography specification using the finite state verification techniques and tools.

As noted before, verification of conformance can be easily performed if the peers communicate in a synchronous fashion. This is because, in this case, system behavior always contains finite number of states (of the order of the product of the number of local states of the peers).

In this paper, we identify the necessary and sufficient conditions under which the behavior of the system when peers communicate asynchronously is *equivalent* to the behavior of the system when peers communicate synchronously, i.e., the peer interactions are synchronizable. If these conditions are satisfied, then choreography conformance of a system can be verified using the system behavior where the peers interact synchronously. We prove that synchronizability can be decided by verifying the equivalence between two variations of the system under consideration. In one variation, the participating peers interact synchronously, while in the other, the peers interact asynchronously and have message

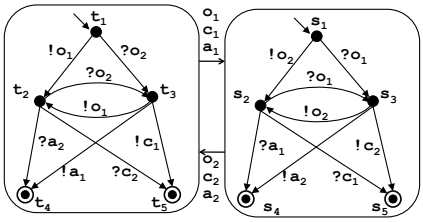


Figure 3: Peers following the Haggle Protocol [15]

queues bounded by 1 (the 1-bounded system). As both variations have finite number of states, the verification of their behavioral equivalence is decidable and can be efficiently performed using existing tools.

Observe that 1-bounded version of System 1 from 1 has a path where the following actions are performed:

$$!r_1\ ?r_1\ !r_1\ !a_1\ ?r_1\ ?a_1\ !a_1\ ?a_1\ !e\ ?e$$

The subsequence containing only the send actions (i.e., the interaction sequence) $!r_1\ !r_1\ !a_1\ !a_1\ !e$ is not present in synchronous version of System 1 (Figure 2(a)). Therefore, System 1 is not synchronizable.

The behavior of System 2, when peers communicate asynchronously, has finite number of states (Figure 2(b)-i). The linear as well as branching behavior of this system w.r.t. send actions is identical to its synchronous counterpart (Figure 2(b)-ii). Using our approach, we can automatically determine that System 2 is synchronizable. Furthermore, based on the synchronizability condition we use, we can also conclude that *any temporal property* that is satisfied by System 2 is also satisfied by its synchronous version, and vice versa.

The behavior of System 3, when peers communicate asynchronously, is shown in Figure 2(c)-i. It has infinite number of states. We prove that System 3 is also synchronizable and any temporal property that is satisfied by System 3 is also satisfied by its synchronous counter-part and vice versa.

Finally, consider the example in Figure 3, where two peers are negotiating on offers following the Haggle protocol [15]. The peers can send an offer (o_i), accept an offer (a_i) or cancel an offer (c_i). Any of the peers can send the first offer. We prove that the system of peers for the Haggle protocol is indeed synchronizable; more specifically, unlike System 2 and System 3, this system is "language" synchronizable. That is, the conformance to choreography specifications expressed as sequences of send actions and/or in Linear Temporal Logic (LTL [6]) can be decided using the system where the peers interact in a synchronous fashion.

In the rest of the paper we discuss different variations of conditions for synchronizability based on the different levels of expressivity of the choreography specification being considered. For instance, if the choreography specification expresses some desired sequencing of send actions (using FSA, regular expressions, LTL) in the peer interactions, then the synchronizability condition is based on language equivalence. We say that systems satisfying this synchronizability condition are language synchronizable. On the other hand, if the choreography specification expresses desired branching behavior (using branching time temporal logics such as CTL, ACTL, CTL* [6]) of the send actions in the peer interactions, then the synchronizability condition is based on bisimulation equivalence or simulation equivalence. Accordingly, we say that systems satisfying these conditions are either bisimulation synchronizable or simulation synchronizable.

In the earlier work on synchronizability [11], only language based choreography conformance was considered and only sufficient (but not necessary) conditions were identified for verifying synchronizability. The sufficient conditions on synchronizability presented in [11], are satisfied by the Systems 2 and 3 in Figure 2; however the satisfiability of these conditions only guarantees that the languages resulting from asynchronous and synchronous interactions of peers are identical. In contrast, our conditions ensure that any temporal property (linear and branching time) satisfied by asynchronous system is also satisfied by the synchronous counter-part. Furthermore, since the conditions given in [11] are sufficient conditions, violation of such conditions may result in false positives, when synchronizable systems do not satisfy the sufficient condition. For instance, the autonomous condition described in [11] (from any local state, a peer can either send or receive messages but not both) is violated by the peers in Figure 3 and generates a false positive. However, our analysis correctly determines that the system is (language) synchronizable.

3. CHOREOGRAPHY CONFORMANCE

In this section, we first formally define the behavior of the peers and the system. We use automata to represent the peer and system behaviors. A state in the automata corresponds to the configuration of the peer or system and labeled transitions denote how the peer or system evolve (after performing certain actions) from one state to another.

Definition 1 (Peer Behavior). *A peer behavior or simply a peer, denoted by $\mathcal{P}$, is a Finite State Automaton (FSA) (M, T, s_0, F, δ) where M is the union of input (M^{in}) and output (M^{out}) message sets, T is the finite set of states, $s_0 \in T$ is the initial state, $F \subseteq T$ is the set of final states, and $\delta \subseteq T \times (M \cup \{\epsilon\}) \times T$ is the transition relation.*

A transition $\tau \in \delta$ can be one of the following three types: (1) a send-transition of the form $(t_1, !m_1, t_2)$ which sends out a message $m_1 \in M^{out}$, (2) a receive-transition of the form $(t_1, ?m_2, t_2)$ which consumes a message $m_2 \in M^{in}$ from its input queue, and (3) an ϵ-transition of the form (t_1, ϵ, t_2). We write $t \xrightarrow{a} t'$ to denote that $(t, a, t') \in \delta$.

Figures 1 and 3 present FSA representations of three different variations of requester and server peers, and a pair of peers participating in an offer negotiation protocol. The start states in each FSA are shown with incoming transitions with no labels and source, and the final states are shown using ⊙. Transitions between any two states are labeled with send and receive actions.

In the following, we will consider systems described using a set of peers, $\langle \mathcal{P}_1, \ldots, \mathcal{P}_n \rangle$, where $\mathcal{P}_i = (M_i, T_i, s_{0i}, F_i, \delta_i)$ and $M_i = M_i^{\text{in}} \cup M_i^{\text{out}}$, such that $\forall i : M_i^{\text{in}} \cap M_i^{\text{out}} = \emptyset$, $\forall i, j : i \neq j \Rightarrow M_i^{\text{in}} \cap M_j^{\text{in}} = M_i^{\text{out}} \cap M_j^{\text{out}} = \emptyset$. All peer pairs in Figures 1 and 3 satisfy the above criteria.

Definition 2 (System Behavior). *A system behavior or simply a system over a set of peers $\langle \mathcal{P}_1, \ldots, \mathcal{P}_n \rangle$, where $\mathcal{P}_i = (M_i, T_i, s_{0i}, F_i, \delta_i)$ and $M_i = M_i^{in} \cup M_i^{out}$, is denoted*

by an automaton (possibly infinite state) $\mathcal{I} = (M, C, c_0, F, \Delta)$ *where*

1. $M = \cup_i M_i$
2. $C \subseteq \mathcal{Q}_1 \times T_1 \times \mathcal{Q}_2 \times T_2 \ldots \mathcal{Q}_n \times T_n$ *such that* $\forall i \in [1..n] : \mathcal{Q}_i \subseteq (M_i^{in})^*$
3. $c_0 \in C$ *such that* $c_0 = (\epsilon, s_{01}, \epsilon, s_{02} \ldots, \epsilon, s_{0n})$
4. $F \subseteq \{\epsilon\} \times F_1 \times \{\epsilon\} \times F_2 \ldots \times \{\epsilon\} \times F_n$*; and*
5. $\Delta \subseteq C \times M \times C$*, and for* $c = (Q_1, t_1, Q_2, t_2, \ldots Q_n, t_n)$ *and* $c' = (Q_1', t_1', Q_2', t_2', \ldots Q_n', t_n')$
   - *(a)* $c \xrightarrow{!m} c' \in \Delta$ *if* $\exists i, j \in [1..n] : m \in M_i^{out} \cap M_j^{in}$,
     - *i.* $t_i \xrightarrow{!m} t_i' \in \delta_i$,
     - *ii.* $Q_j' = Q_j m$,
     - *iii.* $\forall k \in [1..n] : k \neq j \Rightarrow Q_k = Q_k'$ *and*
     - *iv.* $\forall k \in [1..n] : k \neq i \Rightarrow t_k' = t_k$
   - *(b)* $c \xrightarrow{?m} c' \in \Delta$ *if* $\exists i \in [1..n] : m \in M_i^{in}$
     - *i.* $t_i \xrightarrow{?m} t_i' \in \delta_i$,
     - *ii.* $Q_i = m Q_i'$,
     - *iii.* $\forall k \in [1..n] : k \neq i \Rightarrow Q_k = Q_k'$ *and*
     - *iv.* $\forall k \in [1..n] : k \neq i \Rightarrow t_k' = t_k$
   - *(c)* $c \xrightarrow{\epsilon} c' \in \Delta$ *if* $\exists i \in [1..n]$
     - *i.* $t_i \xrightarrow{\epsilon} t_i' \in \delta_i$,
     - *ii.* $\forall k \in [1..n] Q_k = Q_k'$ *and*
     - *iii.* $\forall k \in [1..n] : k \neq i \Rightarrow t_k' = t_k$

The above definition [4, 10] states that a peer can send a message which gets attached to the tail of the queue of the peer capable of receiving the message, while a peer can receive a message if the corresponding message is at the head of its queue (after the receive action is performed the received message is removed from the queue).

Figures 2(a)-i, (b)-i, (c)-i illustrate the system behavior automata for the three requester/server peers from Figure 1. Each state is annotated with the local states of the peers along with the content of their message queues and the transitions between the states follow the definition given above.

Our objective is to verify whether a given system $\mathcal{I}$ *conforms* to a desired choreography specification $\mathcal{C}$. A choreography specification can be described in different languages, e.g., FSA, temporal logic, and accordingly, the semantics of the conformance depends on the semantics of the choreography specification language.

For instance, if $\mathcal{C}$ is described as a finite state machine over the alphabet of send actions, then one can ask whether the sequences of send actions in $\mathcal{I}$ are *identical* to the ones specified by $\mathcal{C}$. This corresponds to the verification of language equivalence (denoted by $\mathcal{L}(\mathcal{I}) = \mathcal{L}(\mathcal{C})$) where, in our case, language of an FSA ($\mathcal{L}(\cdot)$) is the set of sequences of send actions from its start state to a final state.

The choreography specification $\mathcal{C}$ can be described as a temporal property in the language of LTL. The standard LTL semantics defined for a set of infinite sequences over set of atomic propositions can be easily adapted for choreography specification where send actions will be mapped to atomic propositions and finite sequences of send actions will be extended to form infinite sequences by adding an infinite suffix over a special symbol. In this case, the conformance of $\mathcal{I}$ to $\mathcal{C}$ amounts to verifying $\mathcal{I} \models \mathcal{C}$. In essence, the $\models$ relation checks whether the language of $\mathcal{I}$ is a subset of the language of the Büchi automaton [23] representing the semantics of the $\mathcal{C}$ expressed in LTL. Hence, in this case choreography conformance corresponds to language inclusion.

The above problems of conformance verification (against FSA and LTL) are undecidable in general due to the infinite state-space of $\mathcal{I}$ resulting from asynchronous communication among the peers participating in $\mathcal{I}$. The problem of conformance remains undecidable if the choreography specification $\mathcal{C}$ is represented using an FSA over send actions and the conformance demands that any temporal property satisfied by $\mathcal{C}$ should also be satisfied by $\mathcal{I}$. Yet another possibility is to use branching time temporal logic formulas (specified in branching time logics such as Computation Tree Temporal Logic (CTL) or CTL* [6]) as the choreography specification and demand that $\mathcal{I}$ should satisfy the given temporal logic formulas. For these last two variations of conformance, language equivalence or inclusion are not strong enough and stronger notions of equivalence are needed as we show in the following sections.

In this paper, we provide the necessary and sufficient conditions for synchronizability for all these variations of choreography conformance and prove that checking these conditions is decidable. Hence, our results identify a sub-class of peer systems for which choreography conformance is decidable even when unbounded asynchronous communication is used.

4. SYNCHRONIZABILITY

As noted in the previous section, the conformance problem is undecidable due to potentially infinite state-space of the system $\mathcal{I}$. For instance, System 1 in Figure 2(a)-i has an infinite state-space due to the presence of unbounded sequence of requester's send action (e.g., r_1 and r_2) without the corresponding receive action by the server. On the other hand, if the peers exchange messages via synchronous communication (where each send action is synchronized with the corresponding receive action) the behavior of the system has finite state-space as shown in Figures 2(a)-iii, (b)-ii, (c)-ii.

It is well-known that language equivalence and satisfiability of temporal logic properties are decidable when the state machine, under consideration, has finite state-space. Synchronizability analysis identifies whether the asynchronous and synchronous behaviors of the system are "equivalent" with respect to send actions. When a system is synchronizable, conformance of its synchronous behavior to a given choreography specification proves the conformance of its asynchronous behavior.

As we are concerned with choreography specifications expressed as FSAs, LTL, CTL and CTL*, there are different characterizations of synchronizability. In the following, we formally describe these notions and introduce the concepts necessary for proving the decidability of synchronizability problem.

Definition 3 (Synchronous Behavior). *The synchronous system behavior containing a set of peers* $\langle \mathcal{P}_1, \ldots, \mathcal{P}_n \rangle$*, where* $\mathcal{P}_i = (M_i, T_i, s_{0i}, F_i, \delta_i)$ *and* $M_i = M_i^{in} \cup M_i^{out}$*, is denoted by an automaton* $\mathcal{I}_0 = (M, C, c_0, F, \Delta)$ *where*

1. $M = \cup_i M_i$

2. $C \subseteq T_1 \times T_2 \ldots \times T_n$
3. $c_0 \in C$ *such that* $c_0 = (s_{01}, s_{02} \ldots, s_{0n})$
4. $F \subseteq F_1 \times F_2 \ldots \times F_n$; *and*
5. $\Delta \subseteq C \times M \times C$ *and for* $c = (t_1, t_2, \ldots, t_n)$ *and* $c' = (t'_1, t'_2, \ldots, t'_n)$
   (a) $c \xrightarrow{!m} c' \in \Delta$ *if* $\exists i, j \in [1..n] : m \in M_i^{out} \cap M_j^{in}$,
      i. $t_i \xrightarrow{!m} t'_i \in \delta_i$,
      ii. $t_j \xrightarrow{?m} t'_j \in \delta_j$,
      iii. $\forall k \in [1..n] : k \neq i \wedge k \neq j \Rightarrow t'_k = t_k$
   (b) $c \xrightarrow{\epsilon} c' \in \Delta$ *if* $\exists i \in [1..n]$
      i. $t_i \xrightarrow{\epsilon} t'_i \in \delta_i$,
      ii. $\forall k \in [1..n] Q_k = Q'_k$ *and*
      iii. $\forall k \in [1..n] : k \neq i \Rightarrow t'_k = t_k$

Figures 2(a)-ii, 2(b)-ii and 2(c)-iii show the synchronous behavior automata for the three requester/server systems shown in Figure 1.

Definition 4 (Language Equivalence). *The language of a system* $\mathcal{I} = (M, C, c_0, F, \Delta)$*, denoted by* $\mathcal{L}(\mathcal{I})$*, is the set of sequences of send actions on any path from* c_0 *to any state in* F*. Systems* $\mathcal{I}$ *and* $\mathcal{I}'$ *are language equivalent if and only if* $\mathcal{L}(\mathcal{I}) = \mathcal{L}(\mathcal{I}')$.

For example, one of the elements in the language of the automaton corresponding to the asynchronous version of System 2 (Figure 2(b)-i) is $!r_1!a_1!e$.

Definition 5. $\mathcal{I}$ *is said to be language synchronizable if and only if* $\mathcal{L}(\mathcal{I}) = \mathcal{L}(\mathcal{I}_0)$.

If $\mathcal{I}$ is language synchronizable then the sequences of send actions in $\mathcal{I}$ are identical to that specified by a choreography specification $\mathcal{C}$ (defined as an FSA over send actions) if and only if the sequences of send actions in $\mathcal{I}_0$ are identical to that specified by $\mathcal{C}$. Hence, conformance to a choreography $\mathcal{C}$ can be checked on the finite state automaton of $\mathcal{I}_0$ and the result will tell us if $\mathcal{I}$ conforms to $\mathcal{C}$ or not. A similar approach can be used even when $\mathcal{C}$ is specified as an LTL property. This is because, in this case the objective is to verify language inclusion, i.e., whether $\mathcal{L}(\mathcal{I})$ is a subset of the language of the Büchi automaton representing the semantics of $\mathcal{C}$ expressed in LTL. In short, if $\mathcal{C}$ is specified as an FSA or in LTL and $\mathcal{I}$ is language synchronizable, then the conformance verification can be performed using $\mathcal{I}_0$.

However, language inclusion and equivalence are not strong enough for choreography conformance checking via synchronizability if the choreography specification is given as a branching time temporal logic formula. To handle such cases we use a stronger notion of equivalence.

Definition 6 (Bisimulation Equivalence). *Given two systems* $\mathcal{I} = (M, C, c_0, F, \Delta)$ *and* $\mathcal{I}' = (M', C', c'_0, F', \Delta')$ *and two states,* $t \in C$ *and* $t' \in C'$*,* t *and* t' *are bisimulation equivalent, denoted by* $t \approx t'$*, implies*

$$\forall t \xrightarrow{!m} s : \exists t' \xrightarrow{!m} s' : s \approx s' \text{ and}$$
$$\forall t' \xrightarrow{!m} s' : \exists t \xrightarrow{!m} s : s \approx s'$$

In the above, $\xrightarrow{!m}$ *denotes a sequence of transitions containing zero or more transitions over actions in* $\{\epsilon\} \cup M^{in}$ *or* $\{\epsilon\} \cup M'^{in}$ *and a single transition over* $!m$.

Systems $\mathcal{I} = (M, C, c_0, F, \Delta)$ *and* $\mathcal{I}' = (M', C', c'_0, F', \Delta')$ *are said to be bisimulation equivalent, denoted by* $\mathcal{I} \approx \mathcal{I}'$*, if and only if* $c_0 \approx c'_0$.

The above definition is similar to weak bisimulation equivalence [20] of states (systems) where $\Longrightarrow$ corresponds to ϵ-closure transitions.

Definition 7. $\mathcal{I}$ *is said to be bisimulation synchronizable if and only if* $\mathcal{I} \approx \mathcal{I}_0$.

It is well known that bisimulation equivalence preserves all temporal logic properties including branching time temporal logic properties [6]. Hence, if $\mathcal{C}$ is specified in CTL or CTL*, and $\mathcal{I}$ is bisimulation synchronizable, then the conformance verification can be performed using $\mathcal{I}_0$.

Proposition 1. $\mathcal{I}$ *is bisimulation synchronizable implies that* $\mathcal{I}$ *is language synchronizable.*

The above proposition follows from the fact that while bisimulation synchronizability demands the equivalence of branching behavior w.r.t. send actions, language synchronizability only requires the equivalence between sequences of send actions. Hence, bisimulation synchronizability is a more strict notion for synchronizability and therefore enables conformance verification for a richer set of conformance properties (any property expressed in temporal logic, LTL, CTL, CTL*).

Simulation pre-order relation [20] presents the condition under which one system simulates every (send) actions in the other. We will use this concept in Section 5 to order the systems where peers asynchronously communicate using message queues of different sizes/capacities.

Definition 8 (Simulation Pre-order). *Given two systems* $\mathcal{I} = (M, C, c_0, F, \Delta)$ *and* $\mathcal{I}' = (M', C', c'_0, F', \Delta')$ *and two states,* $t \in C$ *and* $t' \in C'$*,* t *simulates* t' *denoted by* $t \prec t'$*, implies*

$$\forall t \xrightarrow{!m} s : \exists t' \xrightarrow{!m} s' : s \prec s'$$

$\mathcal{I}' = (M', C', c'_0, F', \Delta')$ *simulates* $\mathcal{I} = (M, C, c_0, F, \Delta)$*, denoted by* $\mathcal{I} \prec \mathcal{I}'$*, if and only if* $c_0 \prec c'_0$.

5. DECIDING SYNCHRONIZABILITY

In this section, we identify the necessary and sufficient condition for determining synchronizability. The condition involves comparing the behavior of the system using synchronous communication with the one using asynchronous communication with message queue of size equal to 1.

Definition 9 (k-bounded System). *A k-bounded system (denoted by* $\mathcal{I}_k$*) is a system where the length of message queue for any peer is at most k. The description of k-bounded system behavior is, therefore, realized by augmenting condition 5(a) in Definition 2 to include the condition* $|Q_j| < k$*, where* $|Q_j|$ *denotes the length of the queue for peer* j.

Recall that System 2 (Figure 2(b)-i) has the same behavior for any $k > 0$ bound. Figure 4 shows the 1-bounded System

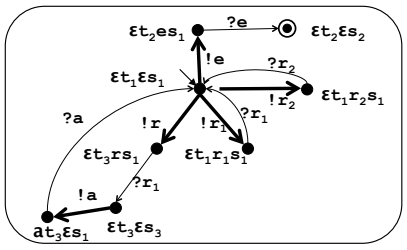


Figure 4: 1-bounded System 3

3; observe that this is "structurally different" from the versions where the peers of System 3 (Figure 1) communicate via pure asynchronous (Figure 2(c)-i) and pure synchronous (Figure 2(c)-ii) interactions.

Proposition 2. $\forall k \geq 0 : [\mathcal{I}_k \prec \mathcal{I}_{k+1} \ \wedge \ \mathcal{L}(\mathcal{I}_k) \subseteq \mathcal{L}(\mathcal{I}_{k+1})]$

PROOF. Every move of a system where peers asynchronously communicate using message queue size k can be simulated (Definition 8) by the system using $> k$ size message queues by avoiding the send actions that makes the receiver's queue size to exceed k. Secondly, $\mathcal{I}_k \prec \mathcal{I}_{k+1}$ implies $\mathcal{L}(\mathcal{I}_k) \subseteq \mathcal{L}(\mathcal{I}_{k+1})$. □

The above proposition along with Definition 5 (or Definition 7) implies that if $\mathcal{I}$ is language (bisimulation) synchronizable then $\forall k \geq 0 : \mathcal{L}(\mathcal{I}_k) = \mathcal{L}(\mathcal{I}_{k+1})$ ($\forall k \geq 0 : \mathcal{I}_k \approx \mathcal{I}_{k+1}$).

Theorem 1.

$$\mathcal{L}(\mathcal{I}_0) = \mathcal{L}(\mathcal{I}_1) \ \Rightarrow \ \forall k \geq 1 : \mathcal{L}(\mathcal{I}_k) = \mathcal{L}(\mathcal{I}_{k+1})$$

PROOF. Consider that there exists $k > 1$ such that $\mathcal{L}(\mathcal{I}_k) \neq \mathcal{L}(\mathcal{I}_1)$. Therefore, there exists a finite path (*witness*) distinguishing $\mathcal{I}_k$ from $\mathcal{I}_1$. In other words, both $\mathcal{I}_k$ and $\mathcal{I}_1$ have a path over the same sequence of send actions such that the path eventually leads to a state from where $\mathcal{I}_k$ can perform a send action which is not possible in $\mathcal{I}_1$. In the following, we will consider paths with $\Longrightarrow$-transitions (see Definition 6).

Consider that such a path with l send actions is

$$t_0^k \stackrel{!m_1}{\Longrightarrow} t_1^k \stackrel{!m_2}{\Longrightarrow} \ldots \stackrel{!m_l}{\Longrightarrow} t_l^k \text{ in } \mathcal{I}_k \tag{1}$$

and the corresponding path in $\mathcal{I}_1$ that deviates from the above after l send actions is

$$t_0^1 \stackrel{!m_1}{\Longrightarrow} t_1^1 \stackrel{!m_2}{\Longrightarrow} \ldots \stackrel{!m_l}{\Longrightarrow} t_l^1 \text{ in } \mathcal{I}_1 \tag{2}$$

such that $\forall j \in [0..l] t_j^k = t_j^1$.

In the above paths, t_l^k is capable of realizing $\stackrel{!m'}{\Longrightarrow}$ which is not possible from t_l^1, i.e., at t_l^1 the peer (say $\mathcal{P}$) which is responsible for consuming m' is not ready to move on any receive action and its message queue is full (contains 1 pending receive action).

As $\mathcal{L}(\mathcal{I}_1) = \mathcal{L}(\mathcal{I}_0)$, there exists a path,

$$t_0^0 \stackrel{!m_1}{\Longrightarrow} t_1^0 \stackrel{!m_2}{\Longrightarrow} \ldots \stackrel{!m_l}{\Longrightarrow} t_l^0 \text{ in } \mathcal{I}_0 \tag{3}$$

We prove by induction that there exists a path over the same sequence of send actions in $\mathcal{I}_1$ where every receives are performed immediately by peer $\mathcal{P}$. Let such a path be

$$t_0'^1 \stackrel{!m'_1}{\Longrightarrow} t_1'^1 \stackrel{!m'_2}{\Longrightarrow} \ldots \stackrel{!m'_l}{\Longrightarrow} t_l'^1 \tag{4}$$

We use $t_j^i{\downarrow_{\mathcal{P}}}$ and $t_j^i{\downarrow_{\mathtt{E}}}$ to denote the local states of the peer $\mathcal{P}$ and the local states of the peers ($\in \mathtt{E}$) other than $\mathcal{P}$ in state t_j^i, respectively. Note that, at states with subscript 0 ($t_0^1, t_0^0, t_0'^1$) are start states in the system, i.e., the local states all peers at these states are identical.

Base case: i=1. If $!m_1$ is an action from some peer in $\mathtt{E}$ to another peer in $\mathtt{E}$, then there exists an identical action $!m_1' = !m_1$ as the states of peers in $\mathtt{E}$ are identical in t_0^1 and $t_0'^1$. The resulting next states of t_0^1 and $t_0'^1$ are also identical.

If $!m_1$ is an action from some peer in $\mathtt{E}$ to the peer $\mathcal{P}$, then there exists an identical action $!m_1' = !m_1$ as the states of peers in $\mathtt{E}$ are identical in t_0^1 and $t_0'^1$. Furthermore, as t_0^0 also allows $!m$, it must have the capability to receive $?m$ at the local state of peer $\mathcal{P}$ in t_0^0. Therefore, the next state of $t_0'^1$ is such that the $t_1'^1{\downarrow_{\mathcal{P}}} = t_1^0{\downarrow_{\mathcal{P}}}$ and $t_1'^1{\downarrow_{\mathtt{E}}} = t_1^1{\downarrow_{\mathtt{E}}}$.

If $!m_1$ is an action from peer $\mathcal{P}$ to some peer in $\mathtt{E}$, then there exists an identical action $!m_1' = !m_1$ as the local state of $\mathcal{P}$ are identical in $t_0'^1$ and t_0^0.

We can, therefore, construct a matching path of length 1 from $t_1'^0$ to $t_1'^1$ such that $t_1'^1{\downarrow_{\mathcal{P}}} = t_1^0{\downarrow_{\mathcal{P}}}$ and $t_1'^1{\downarrow_{\mathtt{E}}} = t_1^1{\downarrow_{\mathtt{E}}}$.

Induction Step. Let $\forall i \leq n : !m_i' = !m_i \wedge t_i'^1{\downarrow_{\mathcal{P}}} = t_i^0{\downarrow_{\mathcal{P}}}$ and $t_i'^1{\downarrow_{\mathtt{E}}} = t_i^1{\downarrow_{\mathtt{E}}}$.

Using the arguments as above, we can prove that $!m_{i+1}' = !m_{i+1}$, and $t_{i+1}'^1{\downarrow_{\mathcal{P}}} = t_{i+1}^0{\downarrow_{\mathcal{P}}}$ and $t_{i+1}'^1{\downarrow_{\mathtt{E}}} = t_{i+1}^1{\downarrow_{\mathtt{E}}}$.

Therefore, paths 2 and 4 are over exactly the same sequence of send actions. Observe that at state $t_l'^1$ peer $\mathcal{P}$ has an empty message queue. As a result $t_l'^1$ is capable of realizing the action $!m'$. This violates our assumption that path 1 is a witness distinguishing $\mathcal{I}_k$ and $\mathcal{I}_1$. □

Theorem 2. $\mathcal{L}(\mathcal{I}_0) = \mathcal{L}(\mathcal{I}_1)$ *if and only if* $\mathcal{I}$ *is language synchronizable.*

PROOF. Follows from Theorem 1 and Propositions 5, 2. □

Systems 2 and 3 (Figure 2(b, c)) are language synchronizable as the language of their respective 1-bounded behavior is identical to that of the corresponding synchronous behavior. Recall that, asynchronous behavior of System 2 is finite state and remains identical for message queues of all sizes. On the other hand, System 3 is infinite state and its behavior "structurally" differs for different message queue size. The 1-bounded system 3 is shown in Figure 4.

The 1-bounded system behavior for the peers (Figure 3) following Haggle protocol is shown in Figure 5(a). In the figure, we have presented the partial view of the system where o_1 is sent before o_2. The dashed arrow corresponds to the case where o_2 is sent followed by o_1 before consuming o_1. The synchronous behavior of the system is shown in Figure 5(b). They are language equivalent as the path (involving the dashed arrow) where $!o_1$ is immediately followed by $!o_2$ without consuming o_1 (and similarly the one where $!o_2$ is immediately followed by $!o_1$), never reaches any of the final states, i.e., these sequences of send actions along these paths are not in the language of the system. Therefore, the system is said to be language synchronizable.

Next, we discuss the conditions under which system is bisimulation synchronizable.

Theorem 3. *The following holds when all peer behaviors in the system are deterministic:*

$$\mathcal{I}_0 \approx \mathcal{I}_1 \Rightarrow \forall k \geq 1 : \mathcal{I}_k \approx \mathcal{I}_{k+1}$$

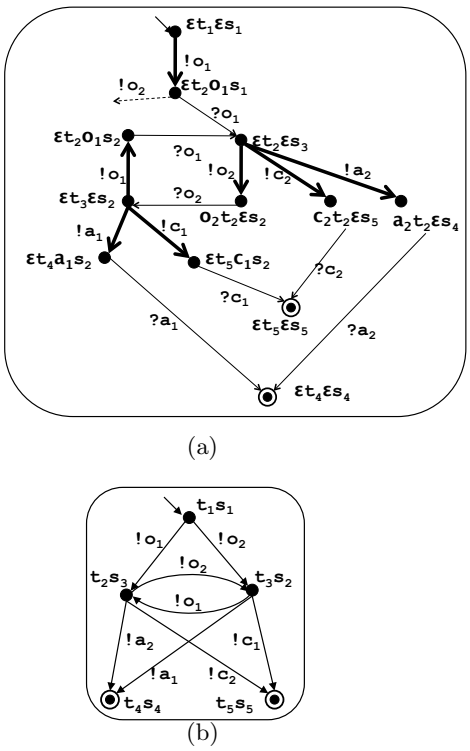


Figure 5: System with peers following Haggle Protocol: (a) 1-bounded; (b) Synchronous

PROOF. Let there exist a $k > 1$ such that $\mathcal{I}_k \not\approx \mathcal{I}_1$. Therefore, there exists a finite path (*witness*) distinguishing $\mathcal{I}_k$ from $\mathcal{I}_1$. In other words, both $\mathcal{I}_k$ and $\mathcal{I}_1$ have a path over the same sequence of states (starting from the corresponding start states) such that the path eventually leads to a state from where $\mathcal{I}_k$ can perform a send action which is not possible in $\mathcal{I}_1$ due to limited size of of peer message queues in the latter.

Consider that such a path with l send actions is

$$t_0^k \stackrel{!m_1}{\Longrightarrow} t_1^k \stackrel{!m_2}{\Longrightarrow} \ldots \stackrel{!m_l}{\Longrightarrow} t_l^k \text{ in } \mathcal{I}_k \tag{5}$$

and the corresponding path in $\mathcal{I}_1$ that deviates from the above after l send actions is

$$t_0^1 \stackrel{!m_1}{\Longrightarrow} t_1^1 \stackrel{!m_2}{\Longrightarrow} \ldots \stackrel{!m_l}{\Longrightarrow} t_l^1 \text{ in } \mathcal{I}_1 \tag{6}$$

such that $\forall j \in [0..l] : t_j^k = t_j^1$.

In the above paths, t_l^k is capable of performing $\stackrel{!m'}{\Longrightarrow}$ which is not possible from t_l^1, i.e., at t_l^1 the peer (say $\mathcal{P}$) which is responsible for consuming m' is not ready to move on any receive action and its message queue is full (contains 1 pending receive action).

As $\mathcal{I}_1 \approx \mathcal{I}_0$, there exists a path,

$$t_0^0 \stackrel{!m_1}{\Longrightarrow} t_1^0 \stackrel{!m_2}{\Longrightarrow} \ldots \stackrel{!m_l}{\Longrightarrow} t_l^0 \text{ in } \mathcal{I}_0 \tag{7}$$

such that $\forall j \geq 0 : t_j^0 \approx t_j^1$.

Let $t_j^i \downarrow_{\mathcal{P}}$ and $t_j^i \downarrow_{\mathtt{E}}$ be the local states of the peer $\mathcal{P}$ and the local states of the peers ($\in \mathtt{E}$) other than $\mathcal{P}$ in state t_j^i, respectively. That is, t_j^i is a tuple $\langle t_j^i \downarrow_{\mathcal{P}}, t_j^i \downarrow_{\mathtt{E}} \rangle$. For ease of explanation, in this notation we are not including the queue contents associated with each peer's local state. Recall that, when $i > 0$, the local state of any peer is associated with the contents of its queue, while for $i = 0$, there is no such queue.

Using paths 6 and 7, we construct a new path

$$t_0'^1 \stackrel{!m_1}{\Longrightarrow} t_1'^1 \stackrel{!m_2}{\Longrightarrow} \ldots \stackrel{!m_l}{\Longrightarrow} t_l'^1 \text{ in } \mathcal{I}_1 \tag{8}$$

such that $\forall j \geq 0 : t_j'^1 \downarrow_{\mathtt{E}} = t_j^1 \downarrow_{\mathtt{E}} \ \wedge\ t_j'^1 \downarrow_{\mathcal{P}} = t_j^0 \downarrow_{\mathcal{P}}$.

Recall that, $t_0^1 \approx t_0^0$, i.e., $\langle t_0^1 \downarrow_{\mathcal{P}}, t_0^1 \downarrow_{\mathtt{E}} \rangle \approx \langle t_0^0 \downarrow_{\mathcal{P}}, t_0^0 \downarrow_{\mathtt{E}} \rangle$.

1. The states $\langle t_j^1 \downarrow_{\mathcal{P}}, t_j^1 \downarrow_{\mathtt{E}} \rangle$ and $\langle t_j'^1 \downarrow_{\mathcal{P}}, t_j'^1 \downarrow_{\mathtt{E}} \rangle$ have same set of send actions from some peer in $\mathtt{E}$ to be consumed by another peer in $\mathtt{E}$. The destination states after such send actions are also identical.

2. The states $\langle t_j^1 \downarrow_{\mathcal{P}}, t_j^1 \downarrow_{\mathtt{E}} \rangle$ and $\langle t_j'^1 \downarrow_{\mathcal{P}}, t_j'^1 \downarrow_{\mathtt{E}} \rangle$ have same set of send actions from some peer in $\mathtt{E}$ to be consumed by the peer $\mathcal{P}$. This is because (a) the local states of $\mathtt{E}$ are identical in $t_j'^1$ and t_j^1, (b) the local state of peer $\mathcal{P}$ is identical in $t_j'^1$ and t_j^0, and (c) $t_j^0 \approx t_j^1$ (implies the action, under consideration, from a peer in $\mathtt{E}$ is consumed immediately by the peer $\mathcal{P}$ in t_j^0). The destination states again maintains the same relationship as the current states.

3. Finally, send actions in $\langle t_j^1 \downarrow_{\mathcal{P}}, t_j^1 \downarrow_{\mathtt{E}} \rangle$ from peer $\mathcal{P}$ to any other peer is also present in $\langle t_j'^1 \downarrow_{\mathcal{P}}, t_j'^1 \downarrow_{\mathtt{E}} \rangle$ as (a) the local state of $\mathcal{P}$ is identical in $t_j'^1$ and t_j^0, and (b) $t_j^0 \approx t_j^1$ (same set of send actions are allowed from $t_j^0 \downarrow_{\mathcal{P}}$ and $t_j^1 \downarrow_{\mathcal{P}}$).

   Conversely, we can show that all send actions from peer $\mathcal{P}$ to any other peer at state $\langle t_j'^1 \downarrow_{\mathcal{P}}, t_j'^1 \downarrow_{\mathtt{E}} \rangle$ are also present in $\langle t_j^1 \downarrow_{\mathcal{P}}, t_j^1 \downarrow_{\mathtt{E}} \rangle$.

   We prove this by contradiction. Assume that peer $\mathcal{P}$ can perform an send action at state $\langle t_j'^1 \downarrow_{\mathcal{P}}, t_j'^1 \downarrow_{\mathtt{E}} \rangle$ and the same action is blocked in $\langle t_j^1 \downarrow_{\mathcal{P}}, t_j^1 \downarrow_{\mathtt{E}} \rangle$ and $\langle t_j^0 \downarrow_{\mathcal{P}}, t_j^0 \downarrow_{\mathtt{E}} \rangle$ (as $t_j^1 \approx t_j^0$). Recall that, we have constrained the the peer behaviors to be deterministic (i.e., $\mathcal{I}_0$ is also deterministic) and $\mathcal{I}_0 \approx \mathcal{I}_1$. This implies one sequence of send actions cannot lead to two different states in the system behavior. Therefore, $\mathcal{I}_0$ is bisimulation equivalent to $\mathcal{I}_1$ implies that t_j^0 and $t_j'^1$ are also bisimulation equivalent. This implies that if t_j^0 any send action from t_j^0 is also possible from $t_j'^1$. This results in contradiction.

Therefore,

$$\forall j \geq 0 : t_j^1 \approx t_j'^1 \tag{9}$$

Observe that peer $\mathcal{P}$'s message queue is empty at state $t_l'^1$ and $t_l'^1 \downarrow_E = t_l^1 \downarrow_E$. As a result, $t_l'^1$ is capable of performing the action $!m'$. Therefore, the assumption that the path 5 is a witness distinguishing $\mathcal{I}_k$ and $\mathcal{I}_1$ does not hold. □

Theorem 4. *The following holds when all peer behaviors in the system are deterministic. $\mathcal{I}_0 \approx \mathcal{I}_1$ if and only if $\mathcal{I}$ is bisimulation synchronizable.*

PROOF. Follows from Theorem 3 and Propositions 7, 2. □

System 2 (Figure 1) is bisimulation synchronizable as the branching behavior of the send actions in the automaton

(synchronous system) in Figure 2(b)-ii is bisimilar to the branching behavior of send actions in the automaton (1-bounded system) in Figure 2(b)-i. Recall that, for any k, the behavior of k-bounded System 2 is exactly identical to 1-bounded System 2.

This is, however, not true for System 3 where k-bounded behavior is structurally different from $k-1$-bounded behavior. In spite of that, the synchronous behavior of System 3 (Figure 2(b)-ii) is bisimilar to 1-bounded System 3 (Figure 4), proving that all k-bounded System 3 are bisimilar. In other words, System 3 is bisimulation synchronizable.

The system with peers following the Haggle protocol (Figure 3) is not bisimulation synchronizable, as its 1-bounded behavior allows o_1 immediately followed by o_2 which can eventually result in peers sending offers even after there is a cancel or an accept message in the peers' message queue. This behavior is not bisimilar to the synchronous behavior of the system (Figure 5(b)). Note that, this behavior of the asynchronous system that witnesses the non-bisimilarity of the 1-bounded system and its synchronous counter-part, never leads to any final states of the system. If the choreography specification is only concerned with the behavior of the system that always eventually leads to the final states of the system, then we can discard the states and the transitions in the 1-bounded system that do not participate in any paths from the start to the final states of the system. The bisimulation equivalence between synchronous and 1-bounded system can be checked after discarding such states and transitions, and the Theorems 3 and 4 remain valid as the definitions of bisimulation equivalence as well as the proofs of the theorems do not depend on the (un)reachability of the final states.

Remark 1. *If the choreography specification is described using universal fragment of CTL, i.e., ACTL, then one can define* simulation synchronizability. *This is because if $\mathcal{I}$ and $\mathcal{I}'$ are simulation equivalent, i.e., $\mathcal{I} \prec \mathcal{I}' \wedge \mathcal{I}' \prec \mathcal{I}$, then $\mathcal{I}$ and $\mathcal{I}'$ conform to the same set of choreography specifications expressed as ACTL properties. The proof, that when peer behaviors are deterministic $\mathcal{I}_0 \prec \mathcal{I}_1 \wedge \mathcal{I}_1 \prec \mathcal{I}_0$ if and only if $\mathcal{I}$ is simulation synchronizable, follows the same arguments as in Theorem 3.*

Synchronizability allows for verifying choreography conformance of $\mathcal{I}$ using $\mathcal{I}_0$. We have presented three different variations of synchronizability: language, bisimulation and simulation synchronizability; each variation corresponds to the expressive power and semantics of the choreography specification language. Bisimulation synchronizability allows for verifying conformance of $\mathcal{I}$ to choreography specification expressed in *any temporal logic*; Simulation synchronizability allows for verifying conformance of $\mathcal{I}$ to choreography specifications expressed in *universal fragment of temporal logic*; and finally, language synchronizability allows for verifying conformance of $\mathcal{I}$ to choreography specification expressed as *FSA and in LTL temporal logic*. Bisimulation synchronizability implies simulation synchronizability which, in turn, implies language synchronizability.

We have proved that language, bisimulation and, respectively, simulation equivalence between $\mathcal{I}_0$ and $\mathcal{I}_1$ is the necessary and sufficient condition for language, bisimulation and simulation synchronizability. As $\mathcal{I}_0$ and $\mathcal{I}_1$ are automata with finite state-space, bisimulation and simulation synchronizability are decidable for systems with deterministic peers, and language synchronizability is decidable for systems with non-deterministic peers. In short, our results identify a subclass of peer systems for which choreography conformance is decidable even when peers interact by exchanging messages asynchronously using unbounded message queues.

6. RELATED WORK

The synchronizability problem was first proposed in [9, 11]. The synchronizability definition used in [9, 11] corresponds to the language-synchronizability definition we use in this paper. Hence it cannot be used to check for conformance of branching time properties. Moreover, the synchronizability conditions given in [9, 11] are sufficient but not necessary conditions. For example, as we discussed earlier in the paper (see Section 2), one of the synchronizability conditions used in [9, 11] is called autonomous condition, and this condition prevents a peer from having a send and a receive transition from the same state. This condition sometimes fails for peer behaviors that are synchronizable, leading to false positives (which is the case for the Haggle protocol shown in Figure 3).

The decidability of synchronizability has been an open problem since it has been defined in [9, 11]. In this paper we show that synchronizability is decidable by giving a computable necessary and sufficient condition for synchronizability. Furthermore, we extend the synchronizability definition to bisimulation synchronizability that enables choreography conformance checking for branching time properties.

In [7], message patterns expressed with Petri nets using synchronous communication are "de-synchronized", i.e., one is interested in finding a specification that produces the same pattern of messages when communications become asynchronous. However, in [7] instead of finding necessary and sufficient conditions for equivalence between the synchronous and asynchronous behavior, the authors try to eliminate race conditions that are created due to asynchronous behavior by several resolution strategies.

In [18] different communication models including synchronous communication and asynchronous communication are defined with the goal of choosing the most appropriate communication model for a given choreography specification. However, one of the assumptions used in [18] limits the behaviors of the analyzed systems to use a finite size message queues. Our results on synchronizability does not require such a restriction and therefore can be used to extend the approach presented in [18] to systems with infinite state spaces.

The work on session types [12, 13] formulates the conformance of an interaction to a predefined choreography protocol as a typing problem. The idea is to first define a global type for interaction behavior which corresponds to the choreography specification. Then during implementation of each peer, it is checked if each local peer implementation is "typable" with respect to the global type. If that is the case then the typing rules ensure that when the peers are executed, they conform to the choreography specification that corresponds to the global type. Interestingly, the type system for session types contains an analogue of the autonomous condition from [9, 11] and therefore cannot be used for choreography conformance checking of some synchronizable peer behaviors.

7. CONCLUSION

Message-based interaction mechanisms are becoming in-

creasingly common in many software infrastructures including service oriented architecture, distributed systems programming, and concurrent programming at the systems level. Message-based communication provides a clean way of isolating behaviors of individual peers that participate in a distributed system. Furthermore, it enables software developers to specify global properties about the interaction behavior among the peers, which is called choreography specification.

In this paper we focused on one of the essential problems in choreography analysis: choreography conformance. In the presence of asynchronous communication, choreography conformance problem becomes undecidable. We showed that for a class of systems choreography conformance can be checked efficiently by replacing the asynchronous communication operations with synchronous communication, which results in a finite state system. The key to this approach is figuring out the cases where replacing asynchronous communication with synchronous communication does not affect the interaction behavior. In this paper, we showed that this problem, called synchronizability, can be solved by comparing the behavior of a system with synchronous communication to the behavior of the same system with bounded asynchronous communication where the queue sizes are limited to one. We also defined different variations of the synchronizability problem for different types of choreography conformance checks and gave necessary and sufficient conditions for each of the variations. Our results are applicable to analysis of the global interaction behavior in any software infrastructure that supports message-based interactions.

8. REFERENCES

[1] J. Armstrong. Getting erlang to talk to the outside world. In *Proceedings of the 2002 ACM SIGPLAN workshop on Erlang*, pages 64–72, 2002.

[2] G. Banavar, T. D. Chandra, R. E. Strom, and D. C. Sturman. A case for message oriented middleware. In *13th International Symposium on Distributed Computing (DISC)*, pages 1–18, 1999.

[3] D. Brand and P. Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, 1983.

[4] T. Bultan, X. Fu, R. Hull, and J. Su. Conversation specification: A new approach to design and analysis of e-service composition. In *Proc. 12th Int. World Wide Web Conf.*, pages 403–410, May 2003.

[5] M. Carbone, K. Honda, N. Yoshida, R. Milner, G. Brown, and S. Ross-Talbot. A theoretical basis of communication-centred concurrent programming.

[6] E. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.

[7] G. Decker, A. P. Barros, F. M. Kraft, and N. Lohmann. Non-desynchronizable service choreographies. In *International Conference on Service Oriented Computing (ICSOC)*, pages 331–346, 2008.

[8] M. Fähndrich, M. Aiken, C. Hawblitzel, O. Hodson, G. C. Hunt, J. R. Larus, and S. Levi. Language support for fast and reliable message-based communication in singularity os. In *Proc. 2006 EuroSys Conf.*, pages 177–190, 2006.

[9] X. Fu, T. Bultan, and J. Su. Analysis of interacting web services. In *Proc. 13th Int. World Wide Web Conf.*, pages 621 – 630, New York, May 2004.

[10] X. Fu, T. Bultan, and J. Su. Conversation protocols: A formalism for specification and analysis of reactive electronic services. *Theoretical Computer Science*, 328(1-2):19–37, November 2004.

[11] X. Fu, T. Bultan, and J. Su. Synchronizability of conversations among web services. *IEEE Trans. Software Eng.*, 31(12):1042–1055, 2005.

[12] K. Honda, V. T. Vasconcelos, and M. Kubo. Language primitives and type discipline for structured communication-based programming. In *7th European Symposium on Programming on Programming Languages and Systems (ESOP'98)*, pages 122–138, 1998.

[13] K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. In *Proc. 35th Symp. on Principles of Programming Languages*, pages 273–284, 2008.

[14] G. C. Hunt and J. R. Larus. Singularity: rethinking the software stack. *Operating Systems Review*, 41(2):37–49, 2007.

[15] Conversation support for agents, e-business, and component integration. http://www.research.ibm.com/convsupport.

[16] Java API for XML messaging (JAXM). http://java.sun.com/developer/earlyAccess/xml/jaxm/.

[17] Java Message Service. http://java.sun.com/products/jms/.

[18] R. Kazhamiakin, M. Pistore, and L. Santuari. Analysis of communication models in web service compositions. In *Proceedings of the 15th international conference on World Wide Web (WWW 2006)*, pages 267–276, 2006.

[19] D. A. Menascé. Mom vs. rpc: Communication models for distributed applications. *IEEE Internet Computing*, 9(2):90–93, 2005.

[20] R. Milner. *Communication and Concurrency*. Prentice-Hall, New York, New York, 1989.

[21] Microsoft Message Queuing Service. http://www.microsoft.com/windowsserver2003/technologies/msmq/default.mspx.

[22] Z. Stengel and T. Bultan. Analyzing singularity channel contracts. In *Proceedings of the 18th International Symposium on Software Testing and Analysis (ISSTA 2009)*, pages 13–24, 2009.

[23] M. Y. Vardi. Automata-theoretic model checking revisited. In *Proceedings of the International Conference on Verification, Model Checking and Abstract Interpretation*, pages 137–150, 2007.

[24] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, and D. F. Ferguson. *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*. Prentice Hall, March 2005.