Choreography Modeling and Analysis with Collaboration Diagrams

Tevfik Bultan
University of California, Santa Barbara
bultan@cs.ucsb.edu

Xiang Fu
Hofstra University
xfu2006@gmail.com

1 Introduction

UML collaboration diagrams (called communication diagrams in [8]) provide a convenient visual model for specifying Web Service choreographies. A choreography specifies the desired set of interactions among a set of Web services. We formalize the interactions among Web services as *conversations*, i.e., the sequence of messages exchanged among the services, recorded in the order they are sent. This paper reviews our recent results on the *realizability problem* for choreographies specified as collaboration diagrams [4, 5]. The realizability problem investigates the following question: Is it possible to construct a set of peers that generate exactly the same set of conversations specified by a given choreography? To study this problem, we model a set of Web services (i.e., peers) as a set of communicating finite state machines [3] and we identify a set of sufficient conditions for realizability of a class of collaboration diagrams.

2 Collaboration Diagrams and Conversations

In a collaboration diagram a set of peers communicate via messages. Each message send event has a unique sequence label. A sequence label consists of a (possibly empty) string of letters (which we call the prefix) followed by a numeric part (which we call the sequence number). The numeric ordering of the sequence numbers defines an implicit total ordering among the message send events with the same prefix. For example, event A2 can occur only after the event A1, but B1 and A2 do not have any implicit ordering. It is also possible to explicitly state dependency relationship among events. For example if an event e is marked with "B2,C3/A2" then A2 is the sequence label of e, and the events with sequence labels B2, C3 and A1 must precede e. In a collaboration diagram we use the notion of *message threads* to refer to a set of messages that have the same prefix (and, therefore, are totally ordered) and that can be interleaved arbitrarily with other messages.

As an example, consider the collaboration diagram in Figure 1 for the Purchase Order Handling service described in the BPEL language specification [2]. All the messages in this example are transmitted asynchronously. There are four threads (the main thread, which corresponds to the empty prefix, and the threads with labels A, B and C). The interactions between the Vendor and the Shipping, Scheduling and Invoicing peers are executed concurrently. However, there are some dependencies among these concurrent interactions: *shipType* message should be sent after the *shipReq* message is sent, the *shipSchedule* message should be sent after the *shipInfo* message is sent, and the *orderReply* message should be sent after all the other messages are sent.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

This work is supported by NSF grants CCF-0614002 and CCF- 0716095.

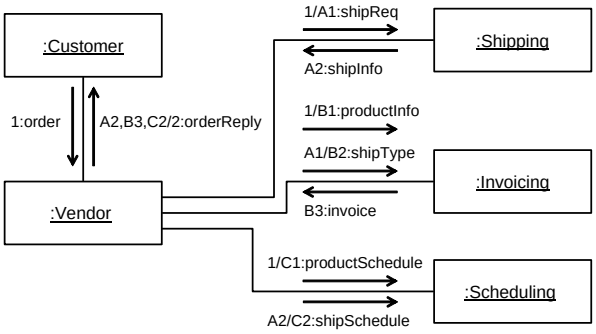


Figure 1: An example collaboration diagram for a composite web service.

Based on the assumptions discussed above we formalize the semantics of collaboration diagrams as follows.

Definition 1: A *collaboration diagram* $\mathcal{D} = (P, L, M, E, D)$ consists of a set of peers P, a set of links $L \in P \times P$, a set of messages M, a set of message send events E, and a dependency relation $D \subseteq E \times E$ among the message send events. Each event has one of the following three recurrence types: 1 (regular), $?$ (conditional), and $*$ (iterative). A dependency $(e_1, e_2) \in D$ means that e_1 has to occur before e_2. We assume that there are no circular dependencies. An event e is an *initial event* of $\mathcal{D}$ if it has no incoming edges in $\mathcal{D}$.

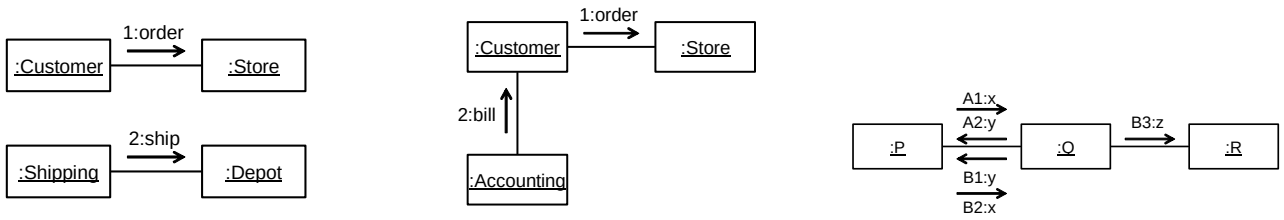
Given a collaboration diagram $\mathcal{D}$ we denote the *set of conversations* defined by $\mathcal{D}$ as $\mathcal{C}(\mathcal{D})$ where $\mathcal{C}(\mathcal{D}) \subseteq M^*$. $\mathcal{C}(\mathcal{D})$ specifies the desired behaviors in a global perspective. A *conversation* $\sigma = m_1 m_2 \dots m_n$ is in $\mathcal{C}(\mathcal{D})$, i.e., $\sigma \in \mathcal{C}(\mathcal{D})$, if and only if $\sigma \in M^*$ and there exists a corresponding matching sequence of message send events $\gamma = e_1 e_2 \dots e_n$ such that (1) each message in the conversation σ is equal to the message of the matching send event in the event sequence γ; and, (2) the ordering of the events in the event sequence γ does not violate the dependencies in D; and, (3) if an event does not appear in the event sequence γ then it must be either a conditional event or an iterative event; and, (4) only iterative events can be repeated in the event sequence γ.

Next, we model the composition of peers [6, 7]. We assume that each finite state machine has a single FIFO input queue for asynchronous messages. A send event for an asynchronous message appends the message to the end of the input queue of the receiver, and a receive event for an asynchronous message removes the message at the head of the input queue of the receiver.

Definition 2: Each peer $\mathcal{A}_i = (M_i, T_i, s_i, F_i, \delta_i)$ is a nondeterministic FSA where $M_i = M_i^A \cup M_i^S$ is the set of messages that are either received or sent by p_i, T_i is the finite set of states, $s_i \in T$ is the initial state, $F_i \subseteq T$ is the set of final states, and $\delta_i \subseteq T_i \times (\{!, ?\} \times M_i \cup \{\epsilon\}) \times T_i$ is the transition relation. A transition $\tau \in \delta_i$ can be one of the following three types: (1) a send-transition of the form $(t_1, !m, t_2)$, and (2) a receive-transition of the form $(t_1, ?m, t_2)$, and (3) an ϵ-transition of the form (t_1, ϵ, t_2).

A *run* of peers is a sequence of actions (as defined above) taken by the peers. A *complete run* is one such that at the end of run each peer is in a final state and each FIFO queue is empty. The corresponding sequence of messages induced from the send events of a run is called a *conversation*. Given a set of peer state machines $\mathcal{A}_1, \dots, \mathcal{A}_n$ we denote the set of conversations generated by them as $\mathcal{C}(\mathcal{A}_1, \dots, \mathcal{A}_n)$. We call a set of peers *well-behaved* if each partial run is a prefix of a complete run (i.e., well-behaved peers never get stuck).

Definition 3: Let $\mathcal{D}$ be a collaboration diagram. We say that the peer state machines $\mathcal{A}_1, \dots, \mathcal{A}_n$ *realize* $\mathcal{D}$ if $\mathcal{C}(\mathcal{A}_1, \dots, \mathcal{A}_n) = \mathcal{C}(\mathcal{D})$. A collaboration diagram $\mathcal{D}$ is *realizable* if there exists a set of well-behaved peer state machines that realize $\mathcal{D}$.



(a) (b) (c)

Figure 2: Unrealizable collaboration diagrams.

Not all collaboration diagrams are realizable. For example, Figure 2(a) shows a simple collaboration diagram that is not realizable. The conversation set specified by this collaboration diagram is $\{$*order ship*$\}$, i.e. this collaboration diagram specifies a single conversation in which, first, the Customer has to send the *order* message to the store, and then the Shipping department has to send the *ship* message to the Depot. However, this conversation set cannot be generated by any implementation of these peers. Any set of peer state machines that generates the conversation "*order ship*" will also generate the conversation "*ship order*". The Shipping department has no way of knowing when the *order* message was sent to the Store, so it may send the *ship* message before the *order* message which will generate the conversation "*ship order*". Since the conversation "*ship order*" is not included in the conversation set of the collaboration diagram shown in Figure 2(a), this collaborations diagram is not realizable. Figure 2(b) and (c) show two other collaboration diagrams that are not realizable.

3 Sufficient Conditions for Realizability

In this section we present sufficient conditions for realizability of collaboration diagrams.

Definition 4: We call a collaboration diagram *separated* if each message appears in the event set of only one thread, i.e., given a separated collaboration diagram $\mathcal{D} = (P, L, M, E, D)$ with k threads, the event set E can be partitioned as $E = \bigcup_{i=1}^{k} E_i$ where E_i is the event set for thread i, $M_i = \{e.m \mid e \in E_i\}$ is the set of messages that appear in the event set E_i and $i \neq j \Rightarrow M_i \cap M_j = \emptyset$.

Note that dependencies among the events of different threads are still allowed in separated collaboration diagrams. The collaboration diagrams in Figure 1, Figure 2(a) and (b), are separated whereas the collaboration diagram in Figure 2(c) is not separated (because message x is involved in two threads A and B). Based on our experience, requiring a collaboration diagram to be separated is not a significant restriction in practice.

Definition 5: We call the event e *well-informed* if one of the following conditions hold: (1) e is an initial event. (2) The immediate predecessor of e is either a synchronous message send event, or if it is not a conditional or iterative send event, then for e to be well-informed, the sender of the message for e has to be either the receiver or the sender of the message for its immediate predecessor. (3) If an immediate predecessor of an event e is either a conditional or an iterative asynchronous message send event, then, to be well-informed, e cannot be a conditional or iterative send event and it must have the same sender and the receiver but a different message than its immediate predecessor.

Theorem 6: A separated collaboration diagram $\mathcal{D}$ is realizable if all the events $e \in E$ are well-informed.

The proof of the above property is given in [5]. Note that, the events with label 2 in Figures 2(a) and (b) are not well-informed. Well-informedness of the events alone does not guarantee realizability of a collaboration

diagram. Consider the unrealizable and un-separated collaboration diagram shown in Figure 2(c). This collaboration diagram has two threads (A and B) and it is not separated since both threads have send events for messages x and y. Note that, although all the events in this collaboration diagram are well-informed, this collaboration diagram is not realizable. The conversation set specified by this collaboration diagram consists of all interleavings of the sequences xy and yxz which is the set $\{xyyxz, xyxyz, xyxzy, yxzxy, yxxzy, yxxyz, yxyxz\}$. However any set of peer state machines that generate this conversation set will either generate the conversation $xyzxy$ or will not be well-behaved. Consider any set of peer state machines that generate this conversation set. Consider the partial run in which first peer P sends x and then the peer Q sends y. From the peer Q's perspective there is no way to tell if y was sent first or if x was sent first. If we require peer Q to receive the message x before sending y (hence, ensuring that x is sent before y) then we cannot generate the conversations that start with the prefix yx. Hence, peer Q can continue execution assuming that the conversation being generated is $yxzxy$ and send the message z before peer P sends another message. Such a partial execution will generate the sequence xyz which is not the prefix of any conversation in the conversation set of the collaboration diagram. Therefore such a partial execution will either lead to a complete run and generate a conversation that is not allowed or it will not lead to any complete run, either of which violate the realizability condition.

4 Conclusion

To the best of our knowledge, realizability of collaboration diagrams has not been studied before our work in [4, 5]. There were similar efforts on Message Sequence Charts (MSCs) [1]. However, as MSCs concentrate on specification of local behaviors, earlier results on realizability of MSCs are not applicable to the realizability of collaboration diagrams. In our earlier work, we have studied the realizability of conversations specified using automata, called *conversation protocols* [6, 7].

Analysis of interactions specified by collaborations diagrams is becoming increasingly important in the web services domain where autonomous peers interact with each other through messages to achieve a common goal. Since such interactions can cross organizational boundaries, it is necessary to focus on specification of interactions rather then the internal structure of individual peers. We argue that collaboration diagrams are a useful visual formalism for specification of interactions among web services. However, specification of interactions from a global perspective inevitably leads to the realizability problem. Our work formalizes the realizability problem for collaboration diagrams and gives sufficient conditions for realizability.

References

[1] R. Alur, K. Etessami, and M. Yannakakis. Realizability and verification of MSC graphs. In *Proc. 28th Int. Colloq. on Automata, Languages, and Programming*, pages 797–808, 2001.

[2] Business process execution language for web services (BPEL), version 1.1. `http://www.ibm.com/developerworks/library/ws-bpel`.

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

[4] T. Bultan and X. Fu. Specification of realizable service conversations using collaboration diagrams. In *Proc. of the IEEE International Conference on Service-Oriented Computing and Applications (SOCA 2007)*, pages 122–132, 2007.

[5] T. Bultan and X. Fu. Specification of realizable service conversations using collaboration diagrams. *Service Oriented Computing and Applications*, 2(1):27–39, 2008.

[6] 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.

[7] X. Fu, T. Bultan, and J. Su. Synchronizability of conversations among web services. *IEEE Transactions on Software Engineering*, 31(12):1042–1055, December 2005.

[8] OMG unified modeling language superstructure, version 2.1.2. `http://ww.uml.org/`, October 2007.