Conversation Protocols: A Formalism for Specification and Verification of Reactive Electronic Services ⋆

Xiang Fu, Tevfik Bultan, Jianwen Su

Department of Computer Science,
University of California at Santa Barbara,
Santa Barbara, CA 93106 U.S.A.

Abstract

This paper focuses on the *realizability* problem of a framework for modeling and specifying the global behaviors of *reactive electronic services* (e-services). In this framework, Web accessible programs (*peers*) communicate by asynchronous message passing, and a virtual global watcher silently listens to the network. The global behavior is characterized by a "conversation", which is the infinite sequence of messages observed by the watcher. We show that given a Büchi automaton specifying the desired set of conversations, called a "conversation protocol", it is possible to realize the protocol using a set of finite state peers if three realizability conditions are satisfied. In particular, the synthesized peers will conform to the protocol by generating only those conversations specified by the protocol. Our results enable a top-down verification strategy where (1) A conversation protocol is specified by a realizable Büchi automaton, (2) The properties of the protocol are verified on the Büchi automaton specification, and (3) The peer implementations are synthesized from the protocol via projection.

Key words: E-Service, Asynchronous Communication, Communicating Finite State Automata, Conversation Protocol, Realizability, Composition, Verification

⋆ The preliminary results from this paper were presented in the Eighth International Conference on Implementation and Application of Automata (CIAA 2003) [15]. Bultan was supported in part by NSF Career award CCR-9984822 and NSF grant CCR-0341365; Fu was partially supported by NSF Career award CCR-9984822, NSF grants IIS-0101134 and CCR-0341365; Su was supported in part by NSF grants IIS-0101134 and IIS-9817432.

Email addresses: `fuxiang@cs.ucsb.edu`, `bultan@cs.ucsb.edu`, `su@cs.ucsb.edu`.

1 Introduction

The use of e-services (i.e., self-contained Web accessible programs and devices) has revolutionized the way that business services are provided and deployed. One recent trend is to provide value added *composite* e-services by integrating existing services available on the Web. However, to make such a "composite paradigm" prevail, one has to first resolve the modeling problem, i.e., how to define the public invocation interface so that individual e-services can be discovered and invoked by others (see [20]).

It has been recognized that the stateless function call models like WSDL [32] are inadequate for describing long running complex services. For example, Graunke et al [17] showed that the lack of stateful coordination of server scripts causes problems in Orbitz reservation system, and similar flaws also exist in many other services like Hertz Rental and Register.com. Emerging standards such as BPML [7], BPEL [6] and WSCI [31] resolve this problem by exposing the control flow skeleton of individual e-services, so that an invoker knows how to interact with a service. In contrast to the process oriented view of BPEL [6], the IBM Conversation Support Project [18] concentrates on the interaction in a peer-to-peer conversation session, and proposes the notion of *conversation policy*. In [9], we generalized the concept of a conversation policy to that of a *conversation specification (protocol)*, which describes the conversation logic globally for any number of peers and for finite length conversations. A conversation protocol can be conveniently captured by a finite state automaton, with the set of messages exchanged among peers as the alphabet. In this paper, we study conversation protocols for *reactive* composite e-services by using Büchi automata to characterize infinite conversations.

Though increasingly many e-service standards [6, 32, 21] have been (and are being) proposed by the industry, many fundamental issues remain unclear [20]. For example, one issue is what the underlying communication model for e-services should be. There has been extensive work on synchronous communication models, for example, CSP [19], I/O automata [24] and interface automata [3]. However, the synchronous communication assumption used in these models, i.e., the assumption that the two communicating processes should execute a send and a corresponding receive action synchronously, is not realistic in an environment like the Internet, where there is no global clock, network delay is significant, and moreover, processes are autonomous and may belong to different enterprises. Although asynchrony can be simulated by introducing explicit queue processes into the synchronous model (as in [16]), the introduction of queue processes inhibits the direct application of finite state verification tools.

In our previous work [9], we developed a framework for modeling composite e-services with asynchronous communication. Under this framework, peers (individual e-service components) communicate via asynchronous messaging and each peer maintains a queue for incoming messages. A virtual global watcher keeps track of messages as they occur, i.e., each sent message is simultaneously written to an input queue and concatenated to the watcher. A central notion of a *conversation*, which is a *finite* sequence of messages observed by the watcher, was studied. This model can be regarded as a theoretical abstraction of the asynchronous message passing mechanisms provided by industry efforts such as Java Message Service [30].

Continuing on the study of conversations, this paper extends the model in [9] by focusing on *reactive* composite e-services, where the global conversation is always *infinite*. (However, some peers may terminate in the composition.) Similar to the results of [9] on finite words, in this paper we show that composition of finite state peers generates non ω-regular languages. In addition, we show that the problem of checking if the composition of finite state peers satisfies an LTL property is undecidable due to the unbounded input queues associated with peers. This motivates a top-down approach to specification of composite e-services. We specify the desired set of global conversations of a composite e-service using a Büchi automaton, which we call a *conversation protocol*. A conversation protocol is realizable if the language it accepts is the set of conversations generated by some composite e-service.

Unfortunately, not every conversation protocol is realizable. In this paper, we present three realizability conditions and show that each conversation protocol satisfying the three conditions is realizable. The first condition is called the *lossless join* condition which requires that the protocol should be complete—when projected to individual peers, the Cartesian product of the projections should be exactly the same as the original protocol. The second condition is the *synchronous compatible* condition which ensures that the protocol does not have "illegal states" as specified in [3]. Finally, the third condition is the *autonomous* condition which implies that at any point in the execution, each peer is determined on the choice to send, or to receive, or to terminate. We show that the LTL properties verified on a realizable conversation protocol will be preserved by its synthesized peers. This result supports a top-down verification strategy.

This paper is organized as follows. Section 2 defines the composite e-service model and in particular the notion of a conversation. Section 3 discusses LTL model checking of composite e-services. Section 4 defines the concept of a conversation protocol and establishes the main results on the three realizability conditions. Finally, Section 5 discusses related work, and Section 6 concludes the paper.

2 A Model for E-Services

We introduce the formal model of composite e-services in this section. In our model, a composite e-service consists of a set of peers where each peer maintains one input queue for incoming messages and may send messages to the input queues of other peers (Fig. 1). A global watcher listens silently to the network and observes the global behavior as a sequence of messages at the times of being sent (i.e. enqueued). The time a peer actually consumes a message from its queue is a local decision by the peer. Such a modeling of the global behavior of a composite e-service (or a distributed system) departs from the traditional approach of focusing on the behaviors of individual peers (or subsystems). In this section, we start with the modeling of individual peers. Then, we move to the definition of composite e-services. Finally, we introduce the notion of global configuration, based on which the concept of a conversation is defined.

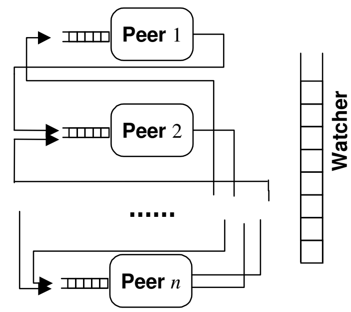

Fig. 1. Composite e-service

For an alphabet Γ, let Γ^*, Γ^ω be the set of all finite, infinite (resp.) words over Γ, and $\Gamma^{\leq\omega} = \Gamma^* \cup \Gamma^\omega$. The definition of a peer is presented as follows.

Definition 1 *A (finite state)* peer *is a nondeterministic Büchi automaton* $(\Sigma^{\text{in}}, \Sigma^{\text{out}}, T, s, F, \Delta)$ *where (1)* Σ^{in} *and* Σ^{out} *are disjoint finite sets of incoming and outgoing (resp.) messages, (2)* T *is a finite set of states,* $s \in T$ *is the initial state, and* $F \subseteq T$ *is a set of final states, (3)* $\Delta \subseteq T \times (\Sigma^{\text{in}} \cup \Sigma^{\text{out}} \cup \{\epsilon\}) \times T$ *is the transition relation (*ϵ *is the empty word).*

A transition is among the following three types: an ϵ-move in the form of (q_1, ϵ, q_2), a receive-transition in the form of $(q_1, ?\alpha, q_2)$, and a send-transition in the form of $(q_1, !\beta, q_2)$, where $\alpha \in \Sigma^{\text{in}}$, $\beta \in \Sigma^{\text{out}}$ and $q_1, q_2 \in T$. A word is *accepted* if some final state is visited infinitely often during the corresponding run. Let $\mathcal{L}(p)$ represent the language accepted by a peer p. Due to the presence of ϵ-moves in a peer p, $\mathcal{L}(p)$ may contain finite words, i.e., $\mathcal{L}(p) \subseteq \Sigma^{\leq\omega}$ where $\Sigma = \Sigma^{\text{in}} \cup \Sigma^{\text{out}}$.

In the following, we define the notion of a "composite e-service" which is the composition of multiple peers. For convenience, we use p_i to denote a peer, and the alphabets, the set of states, etc. of the peer p_i also have subscript i.

Definition 2 *A* composite e-service *is a triple* (n, P, σ) *where*

- $n > 1$ *is an integer,*
- $P = \{p_1, \ldots, p_n\}$ *is a set of* n *peers with pairwise disjoint input alphabets* $\Sigma_1^{\text{in}}, \ldots, \Sigma_n^{\text{in}}$ *and pairwise disjoint output alphabets* $\Sigma_1^{\text{out}}, \ldots, \Sigma_n^{\text{out}}$ *such that* $\cup_i \Sigma_i^{\text{in}} = \cup_i \Sigma_i^{\text{out}}$*, and*
- $\sigma : [1..n] \times (\cup_i \Sigma_i^{\text{out}}) \to [1..n]$ *is a partial mapping such that for each* $i \in [1..n]$*, each* $j \in [1..n]$*, and each* $\beta \in \Sigma_i^{\text{out}} \cap \Sigma_j^{\text{in}}$*,* $\sigma(i, \beta) = j$*.*

Intuitively, $\sigma(i, \beta) = j$ means that the message β can be sent by peer p_i to peer p_j. Note that, based on Definition 2, a message can be transmitted on only one peer to peer channel. Since Definition 1 requires the input and output alphabet of a peer to be disjoint (note that $\Sigma_i^{\text{in}} \cap \Sigma_j^{\text{out}}$ may be nonempty for $i \neq j$), a peer may not send a message back to itself, and hence n is required to be greater than 1 in Definition 2. In the remainder of the paper, we use Σ to denote the entire alphabet of a composite e-service (n, P, σ), i.e., $\Sigma = \bigcup_i \Sigma_i^{\text{in}}$.

Definition 3 *Let $S = (n, \{p_1, \ldots, p_n\}, \sigma)$ be a composite e-service. A* configuration *of S is a $(2n+1)$-tuple of the form $(Q_1, t_1, \ldots, Q_n, t_n, w)$ where for each $j \in [1..n]$, $Q_j \in (\Sigma_j^{\text{in}})^*$ is the queue content of peer p_j, t_j is the state of p_j, $w \in \Sigma^*$ is the message sequence recorded by the global watcher.*

Given two configurations c and c' of a composite e-service $(n, \{p_1, \ldots, p_n\}, \sigma)$, where $c = (Q_1, t_1, \ldots, Q_n, t_n, w)$ and $c' = (Q'_1, t'_1, \ldots, Q'_n, t'_n, w')$, we say that c *derives* c', written as $c \to c'$, if one of the following holds for some $j \in [1..n]$:

- Peer p_j executes an ϵ-move, i.e., $(t_j, \epsilon, t'_j) \in \Delta_j$, and $Q'_j = Q_j$, and $w' = w$, and for each $k \neq j$, $Q'_k = Q_k$ and $t'_k = t_k$, or
- Peer p_j consumes an input, i.e., there exists $\alpha \in \Sigma_j^{\text{in}}$ such that $(t_j, ?\alpha, t'_j) \in \Delta_j$, and $w' = w$, $Q_j = \alpha Q'_j$, and for each $k \neq j$, $Q'_k = Q_k$ and $t'_k = t_k$, or
- Peer p_j sends an output to peer p_i, i.e., there exists $\beta \in \Sigma_j^{\text{out}} \cap \Sigma_i^{\text{in}}$ such that $(t_j, !\beta, t'_j) \in \Delta_j$, and $w' = w\beta$, $Q'_i = Q_i\beta$, and $Q'_l = Q_l$ for each $l \neq i$, and $t'_m = t_m$ for each $m \neq j$.

Next we define the key notions of "run" and "conversation". In the following definition, the watcher content of a configuration $c = (Q_1, t_1, \ldots, Q_n, t_n, w)$, i.e., w, is denoted as $gw(c)$.

Definition 4 *Let $S = (n, \{p_1, \ldots, p_n\}, \sigma)$ be a composite e-service. A* run *γ of S is a finite or infinite sequence of configurations $\gamma = c_0c_1c_2\cdots$ which satisfies the first two of the following conditions, and a* complete run *is an infinite configuration sequence satisfying all of the four conditions.*

(1) $c_0 = (\epsilon, s_1, \ldots, \epsilon, s_n, \epsilon)$ (s_i is the initial state of p_i for each $i \in [1..n]$),
(2) for each $0 \leq i < |\gamma| - 1$, $c_i \to c_{i+1}$,
(3) for each $\ell \in [1..n]$ and each $i \geq 0$, there exist $j > i$ and $k > i$ such that
  (a) t_ℓ^j is a final state, where t_ℓ^j is the state of p_ℓ in c_j, and
  (b) if $Q_\ell^i \neq \epsilon$ then either Q_ℓ^k is empty or $head(Q_\ell^k) \neq head(Q_\ell^i)$, where Q_ℓ^i and Q_ℓ^k are the queue contents of p_ℓ in c_i and c_k respectively.
(4) for each $i \geq 0$, there exists $j > i$ such that $gw(c_i) \neq gw(c_j)$.

An infinite word $w \in \Sigma^\omega$ is a conversation *of S if there exists a complete run $c_0c_1c_2\cdots$ of S such that for each $i \geq 0$, $gw(c_i)$ is a finite prefix of w. Let $\mathcal{C}(S)$ denote the set of conversations of S.*

In Definition 4, Condition (3) requires that during a complete run the Büchi acceptance condition of each peer should be met, and all messages stored in input queues should be eventually consumed; Condition (4) specifies that global message exchange should always eventually advance. The notions of a complete run and a conversation capture the global behaviors where each peer proceeds correctly according to its specification. We call a finite run

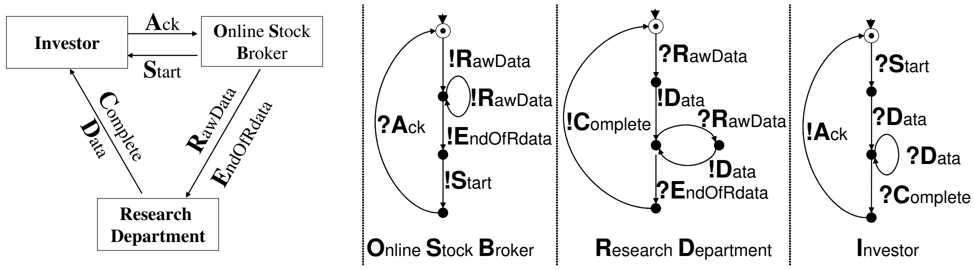

Fig. 2. Fresh Market Update Service

completable if it is the prefix of a complete run. During a complete run, a peer is said to *terminate* at some configuration c_i if after c_i the peer takes ϵ transitions only. During a run, a peer is said to be *receptive* to a message α at configuration c_i if there is a transition starts from the state of the peer at c_i, and consumes α. During a run, a peer is said to be *stuck* at configuration c_i if each transition starting from the state of the peer at c_i is a receive transition, however the peer is not receptive to the message at the input queue head.

3 LTL Model Checking

Given a composite e-service specification, one interesting problem is to check if its conversations satisfy an LTL property. We first define LTL properties [13] on conversations, and then discuss the decidability of LTL model checking.

The set of atomic propositions (AP) of LTL on conversations can be defined as the power set of messages, i.e., $AP = 2^{\Sigma}$. Given a word w written as $w = w_0w_1w_2\ldots$ where w_i is the i-th message in w, w is said to satisfy an atomic proposition ψ, written as $w \models \psi$, if $w_0 \in \psi$. Then the syntax and semantics of LTL formulas on conversations can be naturally extended from the framework defined in [11]. We say that a composite e-service S satisfies an LTL formula ϕ (denoted as $S \models \phi$) if for each conversation $w \in \mathcal{C}(S)$, $w \models \phi$.

One natural question concerning model checking a composite e-service S is whether we can find a Büchi automaton to characterize the conversation set $\mathcal{C}(S)$. A positive answer would imply that many verification techniques become immediately available. Unfortunately, we show that the answer is negative. Consider the composite e-service shown in Fig. 2, which consists of three peers. Each peer is described using a Büchi automaton where "!" and "?" denote a send and receive transition respectively. In each round of message exchange, Online Stock Broker sends a list of "RawData" to Research Department for further analysis, where for each "RawData" one "Data" is generated and sent to Investor. Messages "EndofRdata", "Start", and "Complete" are intended to synchronize the three peers. Finally, Investor acknowledges Online Stock Broker with "Ack" so that a new round of data processing can start. This seemingly simple example produces a non ω-regular set of conversations. Consider its intersection with an ω-regular language $(R^*ESD^*CA)^\omega$ where each message is represented by its initial capital letter. It is easy to infer that the result is $(R^iESD^iCA)^\omega$, because each "Data" sent by Research Department should match a "RawData", and note that during each round all "RawData" are stored in the queue of Research Department before the first "Data" is sent. In addition, "Start" should arrive earlier than "Data", otherwise Investor gets stuck. Using an argument similar to pumping lemma, we can show that this intersection can not be recognized by any Büchi automata.

Proposition 5 *There exists a composite e-service S such that $\mathcal{C}(S)$ is not accepted by any Büchi automaton.*

In fact, given a set of finite state peers, LTL model checking is undecidable. As a two-peer composite e-service is essentially a system of two Communicating Finite State Machines (CFSM) in [8], directly from the CFSM's Turing equivalence result in [8], for each Turing Machine M we can construct a two-peer composite e-service S that simulates M and exchanges a special message (say m_t) once M terminates. Thus M terminates if and only if the LTL formula $S \models \Sigma\ \mathbf{U}\ \{m_t\}$ is true. Here $\mathbf{U}$ is the temporal operator meaning "until" [11], and the meaning of the formula is "eventually message m_t will appear".

Theorem 6 *Given a composite e-service $S = (n, P, \sigma)$ and an LTL property ϕ, determining if $S \models \phi$ is undecidable.*

4 Conversation Protocols

By Proposition 5, the conversation set of an arbitrary composite e-service is not always ω-regular, thus, it is natural to consider a "top-down" approach to composite e-service design by specifying permitted conversations to restrict the global behaviors of a composite e-service. In this section, we introduce the notion of a *conversation protocol* specified using a Büchi automaton. Then we study the concept of *realizability*: given a conversation protocol, is it possible to obtain peers to form a composite e-service which produces exactly the same set of conversations as specified by the protocol? We present three realizability conditions which guarantee that a conversation protocol is realizable.

To facilitate the technical discussions, a *peer prototype* is defined as a pair of disjoint sets $(\Sigma^{\text{in}}, \Sigma^{\text{out}})$. A peer p *implements* a peer prototype $(\Sigma^{\text{in}}, \Sigma^{\text{out}})$ if the input and output alphabets in p are $\Sigma^{\text{in}}, \Sigma^{\text{out}}$ (resp.). Similarly, we can define a *composite e-service prototype* as a composite e-service with peers replaced by peer prototypes, and a composite e-service S implements a composite e-service prototype S^{PROTO} if peers in S implement corresponding peer prototypes in S^{PROTO}. A Büchi automaton $\mathcal{A}$ is *non-redundant* if for every state s in $\mathcal{A}$ there is a run for some accepted word traveling through s.

Definition 7 *Let S^{PROTO} be a composite e-service prototype and Σ be the al-*

phabet of S^{PROTO}. A conversation protocol $\mathcal{A}$ *over* S^{PROTO} *is a non-redundant Büchi automaton with* Σ *as its alphabet.* $\mathcal{A}$ *is* realizable *if there exists a composite e-service* S *which implements* S^{PROTO} *and* $\mathcal{C}(S) = \mathcal{L}(\mathcal{A})$.

Our definition of realizability here is similar to the weak realizability in [4]. A conversation protocol $\mathcal{A}$ satisfies an LTL property ψ, written as $\mathcal{A} \models \psi$ if for all $w \in \mathcal{L}(\mathcal{A})$, $w \models \psi$. The following theorem follows from the well-known results in LTL model checking [13].

Theorem 8 *Given a conversation protocol* $\mathcal{A}$ *and an LTL property* φ, *determining if* $\mathcal{A} \models \varphi$ *is decidable.*

Note that, Theorem 8 does not imply that a top-down approach for composite e-service design is feasible, since not every conversation protocol is realizable.

Example 9 *Consider a composite e-service prototype with four peer prototypes,* p_1, p_2, p_3, p_4, *where* $\Sigma_1^{\text{out}} = \Sigma_2^{\text{in}} = \{\alpha\}$, $\Sigma_3^{\text{out}} = \Sigma_4^{\text{in}} = \{\beta\}$, *and* $\Sigma_1^{\text{in}} = \Sigma_2^{\text{out}} = \Sigma_3^{\text{in}} = \Sigma_4^{\text{out}} = \emptyset$. *The conversation protocol* $\mathcal{A} = (\Sigma, T, s, F, \Delta)$ *with* $\Sigma = \{\alpha, \beta\}$, $T = \{0, 1, 2\}$, $s = 0$, $F = \{2\}$, *and* $\Delta = \{(0, \alpha, 1), (1, \beta, 2), (2, \beta, 2)\}$ *is not realizable, because there is no communication between* p_1 *and* p_3, *hence, there is no way for them to make sure that* α *is sent before any* β *is sent.*

4.1 Characterizing the Conversation Set

In the following, we introduce several interesting tools for the study of conversation set and realizability: (1) a succinct mathematical characterization of the conversation set for each composite e-service, and (2) a powerful tool to prove a conversation protocol is not realizable.

Before the presentation of these results, we need to introduce the notion of *projection* and *join*. Given an alphabet Σ, its subset $\Gamma \subseteq \Sigma$, and a word $w \in \Sigma^{\leq\omega}$, the projection of w onto Γ, written as $\pi_\Gamma(w)$, is a subsequence of w obtained from w by removing all the messages which are not in Γ. The projection to the alphabet Σ_i of a peer prototype p_i is usually written as π_i for simplicity. The projection operator can be naturally extended to a set of words, and the result is the union of the application of projection on each word in the input set. Given a composite e-service of n peers, and n languages $L_1, \ldots, L_n$ where $L_i \subseteq \Sigma_i^{\leq\omega}$ for each $i \in [1..n]$, a *join* operator is defined as:

$$\textsc{join}(L_1, \ldots, L_n) = \{w \mid w \in \Sigma^{\leq\omega}, \forall i \in [1..n] : \pi_i(w) \in L_i\}.$$

It is not hard to infer the following is true:

$$L = \textsc{join}(L_1, \ldots, L_n) \Rightarrow \forall i \in [1..n] : \pi_i(L) \subseteq L_i. \tag{1}$$

Given a conversation protocol $\mathcal{A}$ over composite e-service prototype $S^{\text{PROTO}} = (n, P^{\text{PROTO}}, \delta)$, and a peer prototype $p_i \in P^{\text{PROTO}}$, the *projection* of $\mathcal{A}$ onto p_i, written as $\pi_i(\mathcal{A})$, is a Büchi automaton obtained from $\mathcal{A}$ by replacing each move for a message not in Σ_i by an ϵ-move. We define $S^{\text{PROJ}}(\mathcal{A})$ to be the composite e-service that implements S^{PROTO} and for each $p_i \in P^{\text{PROTO}}$ its implementation $\mathcal{A}_i$ in $S^{\text{PROJ}}(\mathcal{A})$ is $\pi_i(\mathcal{A})$. Based on the above definitions, it can be shown that $\mathcal{L}(\pi_i(\mathcal{A})) = \pi_i(\mathcal{L}(\mathcal{A}))$ for each peer prototype p_i.

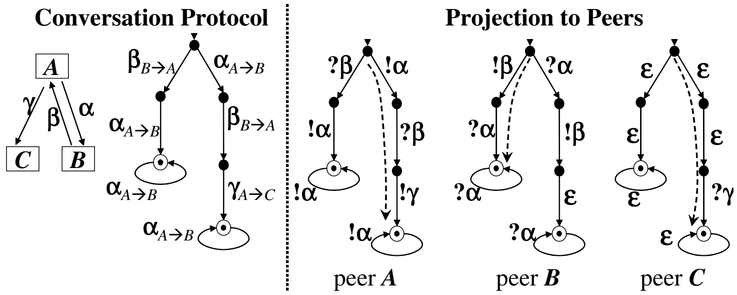

Fig. 3. Ambiguous execution

Example 10 *We present a conversation protocol and its projections to each peer prototype in Fig. 3. The composition of these projections can generate a conversation that does not belong to the original protocol. As indicated by the dotted arrows in Fig. 3, at the beginning, peer B sends a message β to peer A, and β is stored in the input queue of peer A. Then peer A sends message α to peer B, and $\beta\alpha$ is recorded by the global watcher. Now peer B continues to execute the left path of the protocol, consumes the α in the queue; and peer A executes the right path of the protocol, consumes the β, and sends out γ. Eventually, conversation $\beta\alpha\gamma\alpha^\omega$ is generated. Later, we can prove that the conversation protocol in Fig. 3 is not realizable, based on the above argument.*

Let $\mathcal{A}_A$ be the projection of the conversation protocol in Fig. 3 to peer A. Consider the projection of the conversation $\beta\alpha\gamma\alpha^\omega$ to peer A, i.e., $\beta\alpha\gamma\alpha^\omega$. It is not accepted by $\mathcal{A}_A$, however, it is the result of swapping the messages α and β of an accepted word $\alpha\beta\gamma\alpha^\omega$. In fact we can show that the projection of a conversation to each peer is the result of applying zero or more swapping on consecutive output/input message pairs. The reason is that for a peer, an input message can always arrive earlier than it is to be consumed, and be stored in the input queue. To characterize such queuing effect, a *swap closure* operator is defined for each peer prototype.

Let $p_i = (\Sigma_i^{\text{in}}, \Sigma_i^{\text{out}})$ be a peer prototype. Given a word $w \in \Sigma_i^{\leq\omega}$, the *swap closure* of w at peer p_i, written as $\textsc{sc}_i(w)$, is a subset of $2^{\Sigma_i^{\leq\omega}}$ which includes every word w' satisfying the following conditions:

(1) $\pi_{\Sigma_i^{\text{in}}}(w') = \pi_{\Sigma_i^{\text{in}}}(w)$, and
(2) $\pi_{\Sigma_i^{\text{out}}}(w') = \pi_{\Sigma_i^{\text{out}}}(w)$, and
(3) for each integer $j \geq 0$, the number of input messages (of peer p_i) in the prefix $w_0 \dots w_j$ of w is no greater than that of the prefix $w'_0 \dots w'_j$ of w'. (w_j and w'_j are the j-th message in w and w', respectively.)

For example, it is not hard to see that $\textsc{sc}_A(\alpha\beta\gamma\alpha^\omega) = \{\beta\alpha\gamma\alpha^\omega, \alpha\beta\gamma\alpha^\omega\}$, and $\textsc{sc}_A(\beta\alpha\gamma\alpha^\omega) = \{\beta\alpha\gamma\alpha^\omega\}$. $\textsc{sc}_i$ can also be naturally extended to a set of words: given a word set Γ, $\textsc{sc}_i(\Gamma) = \bigcup_{w\in\Gamma} \textsc{sc}_i(w)$. It is not hard to infer that $\textsc{sc}_i$ is idempotent, i.e, for any word set Γ, $\textsc{sc}_i(\textsc{sc}_i(\Gamma)) = \textsc{sc}_i(\Gamma)$. As suggested by the discussion after Example 10, the projection of each conversation to a peer can be regarded as the result of applying $\textsc{sc}_i$ on some accepted word of that projection, we have the following lemma[1].

Lemma 11 *For any composite e-service $S = (n, \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}, \delta)$, its conversation set $\mathcal{C}(S) = \textsc{join}(\textsc{sc}_1(\mathcal{L}(\mathcal{A}_1)), \ldots, \textsc{sc}_n(\mathcal{L}(\mathcal{A}_n)))$.*

Note that Lemma 11 does not imply that analyzing conversation set of an arbitrary composite e-service (e.g. verifying safety properties) is decidable. However, based on Lemma 11 we have the following theorem.

Theorem 12 *A conversation protocol $\mathcal{A}$ is realizable if and only if $\mathcal{L}(\mathcal{A}) = \mathcal{C}(S^{\text{PROJ}}(\mathcal{A}))$.*

Proof: Since $\mathcal{L}(\mathcal{A}) = \mathcal{C}(S^{\text{PROJ}}(\mathcal{A}))$ directly leads to the realizability of $\mathcal{A}$, and $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{C}(S^{\text{PROJ}}(\mathcal{A}))$ is obvious, we only have to prove the following statement: if $\mathcal{A}$ is realizable, then $\mathcal{C}(S^{\text{PROJ}}(\mathcal{A})) \subseteq \mathcal{L}(\mathcal{A})$.

Suppose $\mathcal{A}$ is realized by a composite e-service $S' = (n, \{\mathcal{A}'_1, \ldots, \mathcal{A}'_n\}, \delta)$, by Lemma 11 we have $\mathcal{L}(\mathcal{A}) = \mathcal{C}(S') = \textsc{join}(\textsc{sc}_1(\mathcal{L}(\mathcal{A}'_1)), \ldots, \textsc{sc}_n(\mathcal{L}(\mathcal{A}'_n)))$. Then by Equation 1, for any $i \in [1..n]$,

$$\mathcal{L}(\pi_i(\mathcal{A})) = \pi_i(\mathcal{L}(\mathcal{A})) = \pi_i(\mathcal{C}(S')) \subseteq \textsc{sc}_i(\mathcal{L}(\mathcal{A}'_i)).$$

Apply $\textsc{sc}_i$ on both sides of the above equation, because of the idempotency of $\textsc{sc}_i$, we can infer that $\textsc{sc}_i(\mathcal{L}(\pi_i(\mathcal{A}))) \subseteq \textsc{sc}_i(\mathcal{L}(\mathcal{A}'_i))$. By Lemma 11 this immediately leads to $\mathcal{C}(S^{\text{PROJ}}(\mathcal{A})) \subseteq \mathcal{C}(S')$, which concludes the proof. ▮

Theorem 12 is a powerful tool to prove a conversation protocol is not realizable. For example, by Theorem 12 and the argument in Example 10, the conversation protocol in Fig. 3 is not realizable.

4.2 Realizability conditions

In the following, we present three realizability conditions that guarantee a realizable conversation protocol. We write $w_1 \preceq w_2$ to denote a word w_1 being a prefix of w_2 (w_1 may be equal to w_2). Let $\mathcal{L}^*_{\preceq}(\mathcal{A})$ include all finite prefixes of $\mathcal{L}(\mathcal{A})$ for a Büchi automaton $\mathcal{A}$. Note that $\mathcal{L}^*_{\preceq}(\mathcal{A})$ is regular. For each non-redundant Büchi automaton $\mathcal{A}$ we can construct a standard FSA $\mathcal{A}^*$ by making each state of $\mathcal{A}$ a final state. Obviously $\mathcal{L}(\mathcal{A}^*) = \mathcal{L}^*_{\preceq}(\mathcal{A})$

Lossless join condition Consider the conversation protocol $\mathcal{A}$ in Example 9, $\mathcal{A}$ is not realizable because a conversation $\beta\alpha\beta^\omega$ is not included in $\mathcal{L}(\mathcal{A})$.

[1] A similar result on characterizing the conversation set of a non-reactive composite e-service via the use of join and local prepone operator (which is essentially one single swap on a neighboring output/input message pair) is given in our earlier work [9].

In fact, $\beta\alpha\beta^\omega$ belongs to $\textsc{join}(\pi_1(\mathcal{L}(\mathcal{A})), \pi_2(\mathcal{L}(\mathcal{A})), \pi_3(\mathcal{L}(\mathcal{A})), \pi_4(\mathcal{L}(\mathcal{A})))$. This motivates us to propose a *lossless join* condition to enforce a conversation protocol to be "complete" so that it includes all words in the join of its projections to peer prototypes. For example, when an additional transition $(0, \beta, 0)$ is added into the transition relation of $\mathcal{A}$ in Example 9, it becomes complete (w.r.t. the join of projections) and realizable.

Definition 13 *Let $\mathcal{A}$ be a conversation protocol over a composite e-service prototype $S^{\mathrm{PROTO}} = (n, P^{\mathrm{PROTO}}, \delta)$. $\mathcal{A}$ is* lossless join *if the following is true:*

$$\mathcal{L}(\mathcal{A}) = \textsc{join}(\pi_1(\mathcal{L}(\mathcal{A})), \ldots, \pi_n(\mathcal{L}(\mathcal{A}))).$$

The check of lossless join condition is straightforward. Given a conversation protocol $\mathcal{A}$, construct $S^{\mathrm{PROJ}}(\mathcal{A})$ from $\mathcal{A}$, and then construct the Cartesian product (a generalized Büchi automaton [12] with multiple sets of accepting states) of all peers in $S^{\mathrm{PROJ}}(\mathcal{A})$. Then verify whether the Cartesian product is equivalent to $\mathcal{A}$. Since the Cartesian product may contain ϵ transitions, we have to first check if it accepts words of finite length (note that the original protocol accepts infinite words only). The check is to examine if there is a state from which an infinite ϵ path travels through at least one final state infinitely many times. After confirming the Cartesian product does not accept finite words, the elimination of ϵ-transitions can be achieved by ϵ-transition elimination algorithm for standard Finite State Automata (FSA). Then the Cartesian product can be converted from a generalized Büchi automaton to a standard Büchi automaton, and compared with the original protocol.

Synchronous compatible condition Let us first revisit the Fresh Market Update example presented in Fig. 2. Consider the peer Investor, it is possible that message "Data" arrives earlier than "Start", and then Investor gets stuck. This scenario (under the asynchronous communication assumption) is similar to the case of "illegal state" [3] in the *synchronous composition* of peers. During the synchronous composition of a set of peers, for each message transmitted, its sender and receiver must synchronize their send and receive actions, and hence peers do not need input queues to store incoming messages because they are consumed immediately. The synchronous composition in [3] is actually the Cartesian product of peers in our context. Formally, given a Cartesian product of peers, an *illegal state* is a state in the product where some peer is ready to send out a message but the receiver is not receptive to the message. In the following, we define the *synchronous compatible* condition, to disallow illegal states in the synchronous composition of peers. We will show later, when combined with other conditions, the synchronous compatible condition ensures realizability for conversation protocols. In addition, it prevents peers getting stuck for non-reactive composite e-services (where peers and conversation protocols are described using standard FSA).

Given a composite e-service $S = (n, P, \delta)$, S is said to be *synchronous compat-*

ible if the Cartesian product of P does not have any illegal state. For example, we can make the Fresh Market Update example synchronous compatible by the following changes: (1) introduce an additional message "Ack2" to synchronize the three peers in a lock-step fashion (i.e., for each "Data" received, Investor sends an "Ack2" to Online Stock Broker, and Online Stock Broker does not send out the next "RawData" until the "Ack2" is received), and (2) move the transition for "Start" in Online Stock Broker so that "Start" is sent before the first "RawData" in each round of message exchange.

A conversation protocol $\mathcal{A}$ is synchronous compatible if the "determinized" $S^{\text{PROJ}}(\mathcal{A})$ is synchronous compatible. Formally, the condition is defined as below.

Definition 14 *Let $\mathcal{A}$ be a conversation protocol over $S^{\text{PROTO}} = (n, P, \delta)$, and Σ be the alphabet of S^{PROTO}. $\mathcal{A}$ is said to be* synchronous compatible *if for each word $w \in \Sigma^*$ and each message $\alpha \in \Sigma_a^{\text{out}} \cap \Sigma_b^{\text{in}}$ for $a, b \in [1..n]$, the following holds:*

$$(\forall i \in [1..n],\ \pi_i(w) \in \pi_i(\mathcal{L}^*_{\preceq}(\mathcal{A})))\ \wedge\ \pi_a(w\alpha) \in \pi_a(\mathcal{L}^*_{\preceq}(\mathcal{A}))$$
$$\Rightarrow \quad \pi_b(w\alpha) \in \pi_b(\mathcal{L}^*_{\preceq}(\mathcal{A})).$$

The decision procedure of synchronous compatible condition proceeds as follows: construct $S^{\text{PROJ}}(\mathcal{A})$ from $\mathcal{A}$. Treat every peer in $S^{\text{PROJ}}(\mathcal{A})$ as a standard FSA, and make each state a final state, and then determinize each peer. Construct the Cartesian product of all peers, and check if there is any illegal state. $\mathcal{A}$ is not synchronous compatible if an illegal state is found.

Autonomous condition Synchronous compatible condition is not strong enough to ensure a realizable protocol. It is easy to verify that the protocol in Fig. 3 satisfies both lossless join and synchronous compatible condition. However it is not a realizable protocol.

Take a close look at the execution paths of all peers, which are shown using dotted arrows in Fig. 3. It is clear that the abnormal conversation is the result of "ambiguous" understanding of the protocol by peers, and the racing between A and B at the initial state is the main cause. Consequently, we introduce an *autonomous* condition to restrict racing, so that at any point each peer has exactly one choice: to receive, or to send, or to terminate (unlike peer A can either send out α or receive β at the initial state). Note that here the "determinism" about send/receive/terminate actions does not restrict the nondeterministic decision on which message to send, once the next action is determined to be a send-action.

Let $\mathcal{A}$ be a conversation protocol over $S^{\text{PROTO}} = (n, P^{\text{PROTO}}, \delta)$. For a peer prototype $p_i \in P^{\text{PROTO}}$, we say p_i is *output-ready* (*input-ready*) at a word $w \in \Sigma_i^*$ if there exists a word $w'\alpha \in \mathcal{L}^*_{\preceq}(\mathcal{A})$ such that α is an output (respectively, input) message of p_i and $\pi_i(w') = w$. Similarly p_i is *terminate-ready* at a word $w \in \Sigma_i^*$ if there exists a word $w' \in \mathcal{L}(\mathcal{A})$ such that $\pi_i(w') = w$.

Definition 15 *Let $\mathcal{A}$ be a conversation protocol over a composite e-service prototype $(n, P^{\text{PROTO}}, \delta)$. $\mathcal{A}$ is* autonomous *if for each peer prototype $p_i \in P^{\text{PROTO}}$ and for each finite prefix $w \in \mathcal{L}^*_{\preceq}(\mathcal{A})$, p_i at $\pi_i(w)$ is exactly one of the following: output-ready, input-ready, or terminate-ready.*

Given a conversation protocol $\mathcal{A}$ over $S^{\text{PROTO}} = (n, P^{\text{PROTO}}, \delta)$, we can check the autonomous condition as follows. For each peer prototype $p_i \in P^{\text{PROTO}}$, let $\mathcal{A}_i = (\Sigma_i^{\text{in}}, \Sigma_i^{\text{out}}, T_i, s_i, F_i, \Delta_i)$ be the corresponding peer in $S^{\text{PROJ}}(\mathcal{A})$, and let $T_i' \subseteq T_i$ include each state s where an infinite ϵ path starting at s passes at least one final state for infinitely many times. Construct prefix automaton $\mathcal{A}_i^*$ for each $\mathcal{A}_i$ by making each state in $\mathcal{A}_i$ a final state. Determinize $\mathcal{A}_i^*$ by a standard determinization algorithm for finite state automata. Each state s' of determinized $\mathcal{A}_i^*$ corresponds to a subset of T_i, and we denote it by $T_i(s')$. For each state s', when $T_i(s') \cap T_i'$ is not empty, we require that there is no outgoing transitions starting from s'. If $T_i(s') \cap T_i'$ is empty, then the outgoing transitions from s' are required to be either all send-transitions or all receive-transitions. The complexity of the above check is EXPTIME because of the determinization procedure. The following lemma summarizes the complexity of checking three realizability conditions.

Lemma 16 *Given a conversation protocol $\mathcal{A}$ over a composite e-service prototype S^{PROTO}, whether $\mathcal{A}$ satisfies lossless join, synchronous compatible, and autonomous conditions can be determined in EXPTIME in the size of $\mathcal{A}$ and S^{PROTO}.*

We now proceed to present the main result (Theorem 18), which shows that if the realizability conditions are satisfied, a conversation protocol is realizable.

Lemma 17 *Let $\mathcal{A}$ be a synchronous compatible and autonomous conversation protocol, and $S^{\text{PROJ}}(\mathcal{A})$ be the composite e-service obtained from the projection of $\mathcal{A}$ to each peer, then for each conversation $w \in \mathcal{C}(S^{\text{PROJ}}(\mathcal{A}))$, the following two statements are both true.*

(1) during any complete run of w, each message is consumed eagerly, i.e., a peer never sends nor terminates when its queue is not empty, and
(2) for each peer p_i, $\pi_i(w) \in \pi_i(\mathcal{L}(\mathcal{A}))$.

Proof: Given a complete run γ of a composite e-service $(n, \{p_1, \ldots, p_n\}, \delta)$, if each message is consumed eagerly during γ, i.e., a peer never sends nor terminates when its input queue is not empty, we can infer the following fact: the projection of the conversation generated during γ to each peer is an accepted word of that peer (when that peer is regarded as an automaton that "accepts" words). Based on the above fact, it is not hard to see that statement (1) implies statement (2). Notice that the above property (projection of conversation to a peer is recognized by that peer) may not hold if messages are not consumed eagerly. For example, consider the conversation $\beta\alpha\gamma\alpha^\omega$ generated by peers in

Fig. 3. Its projection to peer A, i.e., $\beta\alpha\gamma\alpha^\omega$, is not an accepted word of peer A. This is because message β is not consumed eagerly by peer A – it is stored in the input queue when peer A sends out α.

Since statement (1) implies statement (2), we concentrate on the proof for statement (1) by contradiction. Assume that for conversation $w = \alpha_0\alpha_1\ldots$ there is a complete run γ where a message α_m from p_x to p_y is the first message that is not consumed eagerly. Since for each $a < m$, α_a is consumed eagerly by its receiver, the projection of word $\alpha_0 \ldots \alpha_{m-1}$ to each peer is the word which represents the path that the peer travels through until the send of α_m. Hence for each peer p_i we have $\pi_i(\alpha_0 \ldots \alpha_{m-1}) \in \pi_i(\mathcal{L}(\mathcal{A}^*))$. We also know that since the input queue of p_x is empty when it sends out α_m, $\pi_x(\alpha_0 \ldots \alpha_m)$ is contained in $\pi_x(\mathcal{L}(\mathcal{A}^*))$. Now, by synchronous compatible condition, $\pi_y(\alpha_0 \ldots \alpha_m)$ should also be contained in $\pi_y(\mathcal{L}(\mathcal{A}^*))$, hence p_y is input ready at word $\pi_y(\alpha_0 \ldots \alpha_{m-1})$.

According to the assumption that message α_m is not consumed eagerly, there are three possibilities: 1) peer p_y sends out a message (let it be α_n) during run γ when α_m is still in its input queue, 2) peer p_y terminates and never consumes α_m, and 3) peer p_y is stuck by message α_m. As Cases 2 and 3 are directly refuted by their conflicts with the definition of a complete run, in the following, we discuss Case 1 only. Consider peer p_y in the run γ. Its local run up to the send of α_n traversed the word $\pi_y(\alpha_0 \ldots \alpha_{m-1}\alpha_n)$ and hence $\pi_y(\alpha_0 \ldots \alpha_{m-1}\alpha_n) \in \pi_y(\mathcal{L}(\mathcal{A}^*))$. Thus peer p_y is output ready at word $\pi_y(\alpha_0 \ldots \alpha_{m-1})$. Combined with the proved fact that p_y is input ready at the same word, this contradicts with the autonomous condition. Therefore the assumption is false, and statement (1) is true. ∎

The statement (2) of Lemma 17 implies the following theorem.

Theorem 18 *A conversation protocol $\mathcal{A}$ is realizable if $\mathcal{A}$ is lossless join, synchronous compatible, and autonomous.*

4.3 Discussion

In the following we discuss several interesting issues about the realizability conditions. First we show that all the three realizability conditions are independent concepts, i.e., neither of them can be expressed as a boolean combination of the other two conditions. Next, we prove that the three conditions are not redundant, in the sense that, for any two of the three realizability conditions, there exists a non-realizable conversation protocol which satisfies those two realizability conditions but not the remaining realizability condition. In other words, missing any one of the realizability conditions will not guarantee realizability. Finally, we show how these conditions fit into practical applications.

In Fig. 4, we show two non-realizable conversation protocols $\mathcal{A}_1$ and $\mathcal{A}_2$. It is

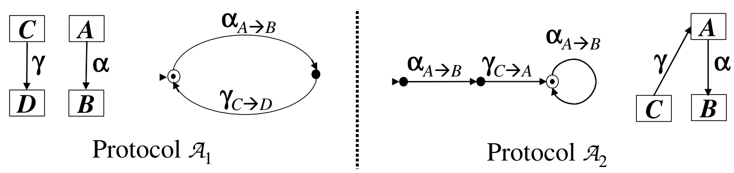

Fig. 4. Examples for Lossless Join and Synchronous Compatible Property

easy to verify that $\mathcal{A}_1$ satisfies autonomous condition and synchronous compatible condition, however it violates the lossless join condition, because word $(\gamma\alpha)^\omega$ is contained in $\text{JOIN}(\pi_A(\mathcal{L}(\mathcal{A}_1)), \ldots, \pi_D(\mathcal{L}(\mathcal{A}_1)))$. Also note that $(\gamma\alpha)^\omega$ will be generated by $S^{\text{PROJ}}(\mathcal{A}_1)$, and by Theorem 12, $\mathcal{A}_1$ is not realizable.

$\mathcal{A}_2$ fails the synchronous compatible condition. The reason is that for empty word ϵ, $\pi_C(\epsilon\gamma)$ is contained in $\pi_C(\mathcal{L}(\mathcal{A}_2^*))$ however $\pi_A(\epsilon\gamma) \notin \pi_A(\mathcal{L}(\mathcal{A}_2^*))$. Concerning the lossless join condition and autonomous condition, it is not hard to see that $\mathcal{A}_2$ satisfies both of them. Because $\gamma\alpha\alpha^\omega$ is a conversation generated by $S^{\text{PROJ}}(\mathcal{A}_2)$, however $\gamma\alpha\alpha^\omega \notin \mathcal{L}(\mathcal{A}_2)$, by Theorem 12, $\mathcal{A}_2$ is not realizable.

As we have shown, the non-realizable conversation protocol in Fig. 3 is one example where synchronous compatible and lossless join conditions are satisfied while autonomous condition is violated. As a summary of the above discussion of Fig. 3 and Fig. 4, we have the following two propositions.

Proposition 19 *Each of the lossless join, synchronous compatible, and autonomous conditions is not equivalent to any boolean combination of the other two conditions.*

Proposition 20 *For each of the lossless join, synchronous compatible and autonomous conditions, there exists a non-realizable conversation protocol which violates that condition while satisfying the other two.*

Now, the next question is, how restrictive are the realizability conditions? We present some facts as a partial answer to this question. It is not hard to see that lossless join condition is a necessary condition for the realizability; however autonomous condition is not. For example, the protocol $\mathcal{A}_3$ in Fig. 5 is realizable but not autonomous. Synchronous compatible condition is not a necessary condition for realizability either. Consider the protocol $\mathcal{A}_4$ shown in Fig. 5. It is not synchronous compatible because at the initial state peer B is not receptive to message γ. However, the protocol is realizable, because $\alpha\gamma\alpha^\omega$ is the only conversation successfully generated by $S^{\text{PROJ}}(\mathcal{A}_4)$. Note that, during the run of $\gamma\alpha^\omega$, peer B is always stuck and hence the word does not count as a conversation. Later we will show that for non-reactive composite e-services we can avoid peers getting stuck. The Proposition 21, given below, summarizes the above discussion.

Proposition 21 *Lossless join is a necessary condition for realizability while autonomous condition and synchronous compatible condition are not.*

The three realizability conditions in Theorem 18 may seem restrictive, how-

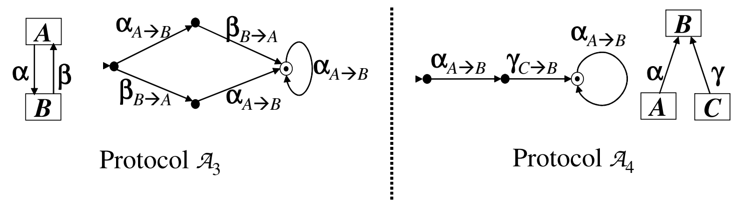

Fig. 5. Two more examples

ever, they are satisfied by many real life e-service applications. We verified that four out of the six examples listed on the IBM Conversation Support site [21] satisfy the conditions, and the other two examples both violate the autonomous condition. For instance, one "Meta Conversation" example in [21], allows the two peers in a meta conversation to race at the beginning to decide who initiates the conversation first. Unfortunately, our autonomous condition forbids such racing. In fact, except restricting the racing between send and receive actions, our realizability conditions allow a certain level of parallelism, which makes it acceptable for many e-services.

The results in this section can be directly applied to the framework in [9], and even better results are achieved. The difference between the two frameworks is that nondeterministic Büchi automata may not be determinized. For a standard FSA conversation protocol $\mathcal{A}$ and its e-service prototype S^{PROTO}, we can determinize each peer implementation in $S^{\text{PROJ}}(\mathcal{A})$, and it is guaranteed that each run in their composition is completable. For example peers can not get stuck by unexpected messages in the composition. For a standard FSA conversation protocol $\mathcal{A}$, $\mathcal{L}(\mathcal{A}) \subseteq \Sigma^*$ and a *complete run* is defined as a finite configuration sequence $\gamma = c_0 c_1 c_2 \cdots c_{|\gamma|-1}$ where in $c_{|\gamma|-1}$ each peer ends up in a final state and each input queue is empty. The extended results are formalized as follows.

Lemma 22 *Let $\mathcal{A}$ be a standard FSA conversation protocol which satisfies synchronous compatible and autonomous conditions. Let $S_D^{\text{PROJ}}(\mathcal{A})$ be the composite e-service obtained by projecting $\mathcal{A}$ to each peer prototype and* determinizing *each projection. Then the following statements are true:*

(1) During any run of $S_D^{\text{PROJ}}(\mathcal{A})$, for each configuration, if the input queue of a peer p_i is not empty and let α be the message at the head of the queue, p_i must be in a non-final state which is receptive to α, and
(2) If both $\mathcal{A}$ and $\mathcal{A}^*$ are lossless join, then every run of $S_D^{\text{PROJ}}(\mathcal{A})$ is completable.

Proof: We assume that γ is the shortest run violating statement (1), and let $w = \alpha_0\alpha_1 \dots \alpha_m$ be the corresponding watcher content. Let p_x, p_y be the sender and receiver of α_m respectively. By a similar argument as of Lemma 17, we can show that p_y is input ready (for message α_m) at $\pi_y(\alpha_0 \dots \alpha_{m-1})$, and $\pi_y(\alpha_0 \dots \alpha_{m-1})$ is the word traversed by the local run of p_y. As p_y is a Deterministic Finite State Automata (DFA), after running $\pi_y(\alpha_0 \dots \alpha_{m-1})$, it advances to a unique state in DFA, and this state is receptive to message α_m.

In addition, by autonomous condition the state is not a final state. Hence the assumption is false and we have proved statement (1).

We provide a constructive proof for statement (2). Given a run γ of $S_D^{\mathrm{PROJ}}(\mathcal{A})$, and w the corresponding watcher content. From statement (1) we can infer the following two facts: (1) by making each peer consume its queue content, γ can be extended to a run γ' so that all queues are empty in the last configuration of γ', and (2) for each peer p_a, $\pi_a(w) \in \pi_a(\mathcal{L}(\mathcal{A}^*))$. Since $\mathcal{A}^*$ is lossless join, w is contained in $\mathcal{L}(\mathcal{A}^*)$. Suppose w is the prefix of a word $w' \in \mathcal{L}(\mathcal{A})$. We can always find a run η for w' s.t. during η each message is consumed immediately after it is sent. Concatenate γ' and the part of η after producing w in the watcher, we get an extension of γ which is a complete run. ∎

5 Related Work

Our model of composite e-services is slightly different than the CFSM model in [8], and almost identical to its variation named Single-Link Communicating Finite State Machines (SLCFSM) [26], except that Büchi automata instead of standard FSA are used in our model. In SLCFSM and our model messages are exchanged through a virtual common medium and stored in the queue associated with the receiver, whereas in [8] each pair of communicating machines use isolated communication channels and each channel has its own queue. The idea of using CFSM with FIFO queues to capture indefinite delay of messages (signals) is similar to many other published models like Codesign Finite State Machine [10], and Kahn Process Networks [22]. Other formalisms like π-Calculus [25] and the recent Microsoft Behave! Project [29] are used to describe concurrent, mobile and asynchronously communicating processes.

Brand and Zafiropulo have shown in [8] that CFSM with perfect FIFO queues are as powerful as Turing Machines. Thus it is not hard to infer that LTL model checking on our e-service composition model is undecidable. This undecidability result is caused by the unbounded FIFO queues, and in [14], many problems are proved to be undecidable even for two identical communicating processes. The transaction sequential consistency problem in [5] provides another perspective for understanding the queue effect, where independent transactions are allowed to commute (which resembles our Prepone operator in [9]). Although FIFO is the most popular assumption about network environment for e-services, industry messaging platforms can provide services with different qualities. For example, Java Message Service [30] allows users to tune the priority, expire date, and persistence of messages to deliver. Undoubtedly, different communication assumptions lead to different analysis complexity for communicating systems. For example, in [2] it is shown that, if perfect FIFO channels are replaced by *lossy* channels, many problems of analyzing CFSM become decidable. As one of our future research plans, it is interesting to study the variations of our current framework with different network assumptions (e.g. FIFO, non-lossy but reordering, and lossy message delivery).

To the best of our knowledge, the notion of realizability on open/concurrent systems was first studied in the late 80's (see [1, 27, 28]). In [1, 27, 28], realizability problem is defined as whether a peer has a strategy to cope with the environment no matter how the environment decides to move. The concept of realizability studied in this paper is rather different. In our model, the environment of an individual peer consists of other peers whose behaviors are also governed by portions of the protocol relevant to them. In addition, our realizability requires that implementation should generate *all* (instead of a subset of) behaviors as specified by the protocol.

A closer notion to "realizability" in this paper is the concept of "weak realizability" of Message Sequence Chart (MSC) Graphs studied in [4]. However, the MSC Graph model captures both "send" and "receive" events, while in our composite e-service model we are interested in the ordering of "send" events only. It can be shown that the MSC Graph model and our conversation protocol model are incomparable in expressiveness. In addition, the three realizability conditions proposed in [4] are different than ours. For example, a conversation protocol which defines the language m^ω does not satisfy the *bounded* condition of [4], but it satisfies the realizability conditions in this paper. Also, it is undecidable to check the other two realizability conditions presented in [4] without the *bounded* condition; while in this paper, each realizability condition can be independently checked in EXPTIME. A notion of *well-formedness* is defined in [26] for SLCFSM, which is a necessary condition for free of deadlocks and free of unspecified receptions. The realizability conditions proposed in this paper are sufficient conditions for realizability, and for standard FSA conversation protocols, these conditions are the sufficient conditions to guarantee free of deadlocks as well. It is interesting to note that state space reduction techniques such as fair reachability analysis [23] for CFSM have a similar idea to balance execution steps among machines, and there are sufficient and necessary conditions to identify finite fair reachable state space so that the detection of some specific logic errors such as deadlock and unspecified receptions is decidable. Our results differ from the fair reachability analysis in that we support general LTL model checking, and allow arbitrary interconnection patterns among peers (fair reachability analysis in [23] requires a cyclic shape of interconnection).

6 Conclusions

In this paper, we studied the global behaviors of reactive composite e-services in terms of the conversations they generate. LTL model checking on composite e-services specified using the bottom-up approach was shown to be undecidable. This suggests that specifying the permitted conversations as conversation protocols in a top-down fashion is beneficial. However, not every conversation protocol defined by a Büchi automaton is realizable by asynchronous peers. We presented three conditions on conversation protocols which ensure realizability. Studying the global behavior of composite e-services is a promising research topic. The results in [9] and in this paper provide a starting point. While our current model is a *static* model, many real-world composite e-services can have dynamic behaviors, for example, dynamically instantiating a new business process and dynamically establishing a new channel with another peer. Extending our model to address these issues is interesting for our future research.

References

[1] M. Abadi, L. Lamport, and P. Wolper. Realizable and unrealizable specifications of reactive systems. In *Proc. of 16th Int. Colloq. on Automata, Languages and Programming*, volume 372 of *LNCS*, pages 1–17. Springer Verlag, 1989.

[2] P. Abdulla and B. Jonsson. Verifying programs with unreliable channels. *Information and Computation*, 127(2):91–101, 1996.

[3] L. D. Alfaro and T. A. Henzinger. Interface automata. In *Proc. 9th Annual Symp. on Foundations of Software Engineering*, pages 109–120, 2001.

[4] R. Alur, K. Etessami, and M. Yannakakis. Realizability and verification of MSC graphs. In *Proc. 28th Int. Colloq. on Automata, Languages, and Programming*, volume 2076 of *LNCS*, pages 797 – 808, 2001.

[5] R. Alur, K. McMillan, and D. Peled. Model-checking of correctness conditions for concurrent objects. *Information and Computation*, 160:167–188, 2000.

[6] Business Process Execution Language for Web Services (BPEL), Version 1.1. `http://www.ibm.com/developerworks/library/ws-bpel`, May 2003.

[7] Business Process Modeling Language (BPML). `http://www.bpmi.org`.

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

[9] 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, 2003.

[10] M. Chiodo, P. Giusto, A. Jurecska, L. Lavagno, H. Hsieh, and A. Sangiovanni-Vincentelli. A formal specification model for hardware/software codesign. In *Proc. of the Intl. Workshop on Hardware-Software Codesign*, October 1993.

[11] E.M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 2000.

[12] C. Courcoubetis, M. Vardi, P. Wolper, and M. Yannakakis. Memory-efficient algorithms for the verification of temporal properties. *Formal Methods in System Design*, 1:275 – 288, 1992.

[13] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science: Volume B: Formal Models and Semantics*, pages 995–1072. Elsevier, 1990.

[14] A. Finkel and P. McKenzie. Verifying identical communicating processes is undecidable. *Theoretical Computer Science*, 174(1–2):217–230, 1997.

[15] X. Fu, T. Bultan, and J. Su. Conversation protocols: A formalism for specification and verification of reactive electronic services. In *Proc. 8th International Conference on Implementation and Application of Automata*, volume 2759 of *LNCS*, pages 188–200, July 2003.

[16] S. J. Garland and N. Lynch. Using I/O automata for developing distributed systems. In *Foundations of Component-Based Systems*. Cambridge Univ. Press, 2000.

[17] P. Graunke, R. B. Findler, S. Krishnamurthi, and M. Felleisen. Modeling web interactions. In *Proc. of 12th European Symp. on Programming*, volume 2618 of *LNCS*, pages 238 – 252, 2003.

[18] J. E. Hanson, P. Nandi, and S. Kumaran. Conversation support for business process integration. In *Proc. of 6th IEEE Int. Enterprise Distributed Object Computing Conference*, 2002.

[19] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.

[20] R. Hull, M. Benedikt, C. Christophides, and J. Su. E-services: A look behind the curtain. In *Proc. 22nd ACM Symp. on Principles of Database Systems*, pages 1 – 14, 2003.

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

[22] G. Kahn. The semantics of a simple language for parallel programming. In *Proc. of IFIP 74*, pages 471 – 475. North-Holland, 1974.

[23] H. Liu and R. E. Miller. Generalized fair reachability analysis for cyclic protocols. In *IEEE/ACM Transactions on Networking*, pages 192–204, 1996.

[24] N. Lynch and M. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proc. 6th ACM Symp. Principles of Distributed Computing*, pages 137–151, 1987.

[25] R. Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999.

[26] W. Peng. Single-link and time communicating finite state machines. In *Proc. of 2nd Int. Conf. on Network Protocols*, pages 126–133, 1994.

[27] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. of 16th ACM Symp. Principles of Programming Languages*, pages 179–190, 1989.

[28] A. Pnueli and R. Rosner. On the synthesis of an asynchronous reactive module. In *Proc. of 16th Int. Colloq. on Automata, Languages, and Programs*, volume 372 of *LNCS*, pages 652–671, 1989.

[29] S. K. Rajamani and J. Rehof. A behavioral module system for the pi-calculus. In *Proc. of Static Analysis Symposium (SAS)*, volume 2126 of *LNCS*, pages 375 – 394, 2001.

[30] Sun. Java Message Service. `http://java.sun.com/products/jms/`.

[31] Web Service Choreography Interface (WSCI) 1.0. `http://www.w3.org/TR/2002/NOTE-wsci-20020808/`, August 2002.

[32] Web Services Description Language (WSDL) 1.1. `http://www.w3.org/TR/wsdl`, March 2001.