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,
Santa Barbara, CA 93106, USA.
{fuxiang,bultan,su}@cs.ucsb.edu

Abstract. This paper focuses on the *realizability* problem of a framework for modeling and specifying the global behavior of *reactive electronic services* (e-services). In this framework, Web accessible programs (*peers*) communicate by asynchronous message passing, and a virtual global watcher listens silently 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 implement it 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, (3) The peer implementations are synthesized from the protocol via projection.

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 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 [18]).

It has been realized that stateless function call models like WSDL [29] are not adequate to describe long running complex services. Indeed [15] shows that lack of stateful coordination of server scripts caused problems in Orbitz reservation, and similar flaws are also observed in many other services like Hertz Rental and Register.com. Emerging standards like BPML [7], BPEL4WS [6] and WSCI [28] can resolve this problem by exposing the abstract control flow skeleton of business processes so that invoker knows how to interact. In contrast to the process oriented view of BPEL4WS [6], IBM Conversation Support Project [16] concentrates on the interaction in a peer-to-peer conversation session, and proposes the notion of *conversation policy*. In [9] we generalized the idea of conversation policy to *conversation specification (protocol)*, which describes conversation logic globally for any number of peers. A conversation protocol can be conveniently captured by a finite state automaton, with the set of messages exchanged among peers as the alphabet. A *reactive* version for e-services would simply use a Büchi automaton, which is a successful methodology for expressing liveness requirements.

Though increasingly many e-service standards [29, 6, 19] have been and are being proposed by the industry, many fundamental issues remain unclear [18]. 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 [17], I/O automata [22] and interface automata [3]. However, their synchronous assumption, i.e., two communicating processes should execute a send and a corresponding receive action synchronously, is not realistic in an environment like Internet, where there is no global clock and network delay is significant. Although asynchrony can be simulated by introducing explicit queue processes in synchronous model, like [14], the introduction of queue processes inhibits the direct application of finite state model checking tools.

In our previous work [9], a framework was developed for modeling e-services with asynchrony assumptions. Under this framework, peers (individual e-service components) communicate via asynchronous messages 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 many industry efforts like Java Message Service [27].

Continuing on the study of e-service conversations, this paper extends the model in [9] by focusing on *reactive* 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 our top-down approach to specification of composite e-services. We specify the desired set of global conversations of an e-service using a Büchi automaton, and we call it a *conversation protocol*.

Unfortunately, not all conversation protocols are realizable. We present three realizability conditions in this paper and show that any conversation protocol which satisfies these three properties is realizable. The first property is called *lossless join* property 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 property is the *synchronous compatible* property which ensures that the protocol does not have "illegal states" as specified in [3]. Finally the third condition is the *autonomous property* which implies that at any point in the execution, each peer, independently, can make a decision on whether to send, or to wait for a message, or to terminate. LTL properties verified on a realizable conversation protocol will be preserved by its synthesized peers, and this result supports a top-down verification strategy.

Related Work Our model of composite e-services is different than the Communicating Finite State Machines (CFSM) model in [8]. In our model message 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 [20]. Other formalisms like π-Calculus [23] and the recent Microsoft Behave! Project [26] 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 [13], 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]). In [2] it is shown that, if perfect FIFO channels are replaced by *lossy* channels, many problems become decidable. However we stick with the perfect FIFO in our model, since we assume that underlying communication protocols (like TCP/IP) ensure perfect FIFO message deliveries.

To the best of our knowledge, the notion of realizability on open/concurrent systems was first studied in the late 80's (see [1, 24, 25]). In [1, 24, 25], 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. A related notion is the realizability of Message Sequence Chart (MSC) Graphs [4]. However, the MSC Graph model captures both "send" and "receive" events, while in our e-composition 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. It is interesting to note that there are other works like fair reachability analysis in [21] which achieved decidable analysis results by restricting both the shape of composition (cyclic connection in [21]) and the control flow of protocol itself.

This paper is organized as follows. §2 defines the e-service model and in particular the notion of an e-service conversation. §3 discusses LTL model checking of e-services. §4 defines the concept of a conversation protocol and establishes the main results on the three realizability conditions. Finally §5 presents our conclusions.

2 A Model for E-Services

We introduce the formal model of composite e-services in this section. In our model (see Fig. 1), an e-service consists of a set of peers where each peer maintains one input queue for incoming messages and may send messages to the queues of other peers. 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 an 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.

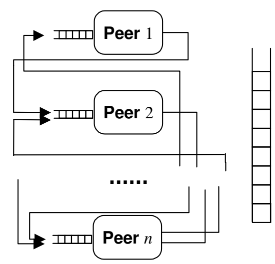

Fig. 1. e-service model

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* peer *is modeled using 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 either an ϵ-move, or it either consumes an incoming message (from the input queue) or produces an output but not both. These three types of moves are denoted by triples (q_1, ϵ, q_2), $(q_1, ?a, q_2)$, $(q_1, !b, q_3)$ respectively. Let $\mathcal{L}(p)$ represent the language accepted by a peer p. (A word is accepted if some final states are visited infinitely often.) 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 an "e-service" which involves a set of 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. *An* e-service *is a triple* (n, P, σ) *where*

- *$n > 1$ is an integer,*
- *$P = \{p_1, p_2, ..., p_n\}$ is a set of n peers with pairwise disjoint input alphabets $\Sigma_1^{\text{in}}, ..., \Sigma_n^{\text{in}}$ and pairwise disjoint output alphabets $\Sigma_1^{\text{out}}, ..., \Sigma_n^{\text{out}}$ (note that $\Sigma_i^{\text{in}} \cap \Sigma_j^{\text{out}}$ may be nonempty for $i \neq j$) 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 $i \in [1..n]$ and each $b \in \Sigma_i^{\text{out}}$, $\sigma(i, b)$ is defined, and*
  - *For each $i \in [1..n]$, each $j \in [1..n]$, and each $b \in \Sigma_i^{\text{out}} \cap \Sigma_j^{\text{in}}$, $\sigma(i, b) = j$.*

Intuitively, $\sigma(i, a) = j$ means that the message a can be sent by peer p_i to peer p_j. Note that in Definition 2 a message can be transmitted on one channel only. In the remainder of the paper, we use Σ to denote the entire set of alphabet of an e-service (n, P, σ), i.e., $\Sigma = \bigcup_i \Sigma_i^{\text{in}}$.

Definition 3. *Let $S = (n, \{p_1, ..., p_n\}, \sigma)$ be an e-service. A* configuration *of S is a $(2n + 1)$-tuple of the form $(Q_1, t_1, ..., Q_n, t_n, w)$ 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 global watcher which records the sequence of messages that have been transmitted.*

For two configurations $\gamma = (Q_1, t_1, ..., Q_n, t_n, w)$, $\gamma' = (Q'_1, t'_1, ..., Q'_n, t'_n, w')$, we say that γ *derives* γ', written as $\gamma \to \gamma'$, if one of the following holds:

- A peer p_j executes an ϵ-move, i.e., there exists $j \in [1..n]$ s.t. $(t_j, \epsilon, t'_j) \in \Delta_j$, and $Q'_j = Q_j$, and $w' = w$, and for each $k \neq j$, $Q'_k = Q_k$ and $t'_k = t_k$.
- A peer p_j consumes an input, i.e., there exist $j \in [1..n]$ and $a \in \Sigma_j^{\text{in}}$ s.t. $(t_j, ?a, t'_j) \in \Delta_j$, and $w' = w$, $Q_j = aQ'_j$, and for each $k \neq j$, $Q'_k = Q_k$ and $t'_k = t_k$.
- A peer p_j sends an output to peer p_k, i.e., there exist $j, k \in [1..n]$ and $b \in \Sigma_j^{\text{out}} \cap \Sigma_k^{\text{in}}$ s.t. $(t_j, !b, t'_j) \in \Delta_j$, and $w' = wb$, $Q'_k = Q_k b$, and $Q'_l = Q_l$ for each $l \neq k$, and $t'_m = t_m$ for each $m \neq j$.

For each configuration $c = (Q_1, t_1, ..., Q_n, t_n, w)$, we denote its global watcher content as $gw(c) = w$. Next we define the key notion of "run" and "conversation".

Definition 4. *Let $S = (n, \{p_1, ..., p_n\}, \sigma)$ be an e-service. A* run *γ of S is a finite or infinite sequence of configurations $\gamma = c_0, c_1, c_2, \ldots$ satisfying the first two of the following conditions, and a* complete run *is an infinite configuration sequence satisfying all of them.*

1. *$c_0 = (\epsilon, s_1, ..., \epsilon, s_n, \epsilon)$ (s_i is the initial state of p_i for each $i \in [1..n]$),*
2. *for each $0 \leq i < |\gamma|$, $c_i \to c_{i+1}$,*
3. *for each $\ell \in [1..n]$ and each $i \geq 0$, there exist $j > i, k > i$ such that*
   (a) t_ℓ^j is a final state, where t_ℓ^j is the state of p_ℓ in c_j.
   (b) $head(Q_\ell^k) \neq head(Q_\ell^i)$ if $Q_\ell^i \neq \epsilon$, 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 a $j > i$ 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 $c_0, c_1, c_2, \ldots$ 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 ever sent should be eventually consumed; condition 4 specifies that global message exchange should eventually advance. The notion of conversation captures the global behaviors where each peer proceeds correctly according to its specification. Note that there might be bad runs where some peer is blocked by an unexpected message, or all peers stay in a waiting (deadlock) state.

3 LTL Model Checking

Given an e-service specification, one interesting problem is to check if its conversations satisfy an LTL property. We will first define LTL properties[12, 11] on conversations. Then the decidability of LTL model checking will be discussed.

For a conversation $w = w_0, w_1, w_2, \dots$, let w_i denote the i-th message in w, and $w^i = w_i, w_{i+1}, w_{i+2}, \dots$ the i-th suffix of w. The set of atomic propositions (AP) is the power set of messages, i.e., $AP = 2^{\Sigma}$. The syntax and semantics of LTL formulas are defined as follows, where $\psi \in AP$ is an atomic proposition, and ϕ and φ are two LTL formulas.

$$
\begin{array}{lll}
w \models \psi & \text{iff} & w_0 \in \psi \\
w \models \neg\phi & \text{iff} & w \not\models \phi \\
w \models \phi \wedge \varphi & \text{iff} & w \models \phi \text{ and } w \models \varphi \\
w \models \phi \vee \varphi & \text{iff} & w \models \phi \text{ or } w \models \varphi \\
w \models X\phi & \text{iff} & w^1 \models \phi \\
w \models G\phi & \text{iff} & \text{for all } i \geq 0, w^i \models \phi \\
w \models \phi U \varphi & \text{iff} & \text{there exists } j \geq 0, \text{ s.t. } w^j \models \varphi \text{ and, for all } 0 \leq i < j, w^i \models \phi
\end{array}
$$

We say that an 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 an 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 system shown in Fig. 2. 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. Message classes "EndofRdata", "Start", and "Complete"

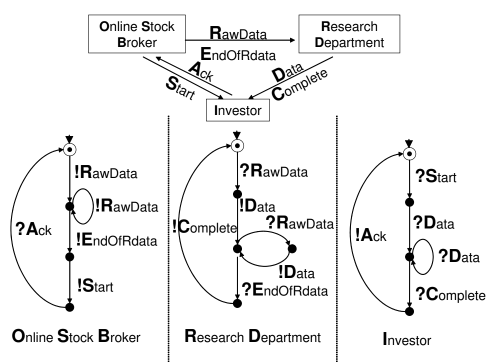

Fig. 2. Fresh Market Update Service

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 $(\mathbf{R}^*\mathbf{ESD}^*\mathbf{CA})^\omega$ (each message is represented by its first letter). It is easy to infer that the result is $(\mathbf{R}^i\mathbf{ESD}^i\mathbf{CA})^\omega$, because Investor enforces that "Start" should arrive earlier than "Data". By an argument similar to pumping lemma, we can show that this intersection cannot be recognized by any Büchi automata.

Proposition 5. *There exists an 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. The proof can be shown by reduction from the halting problem of Turing Machines. For each Turing Machine M we can construct a two-peer e-service S that simulates M and exchanges a special message (say m_t) once M terminates. Thus M terminates if and only if $S \models \Sigma\ U\ m_t$.

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

4 Conversation Protocols

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

To facilitate the technical discussions below, 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 an e-service prototype as an e-service with peers replaced by peer prototypes, and an e-service S implements an e-service prototype S^P if peers in S implement corresponding peer prototypes in S^P. A standard Büchi automaton $\mathcal{A}$ is *nonredundant* if for every state s in $\mathcal{A}$ there is a run of some accepted word traveling through s.

Definition 7. *Let $S^P = (n, P^P, \sigma)$ be an e-service prototype and Σ be the union of all alphabets in S^P. A* conversation protocol *over S^P is a nonredundant Büchi automaton $\mathcal{A} = (\Sigma, T, s, F, \Delta)$ with the alphabet Σ. A conversation protocol $\mathcal{A}$ is* realizable *if there exists an e-service S which implements S^P and $\mathcal{C}(S) = \mathcal{L}(\mathcal{A})$.*

Our definition of realizability here is similar to the weak realizability in [4], where deadlock is not considered. 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-know results in LTL model checking [12, 11].

Theorem 8. *Given a conversation protocol $\mathcal{A}$ for an e-service prototype $S^P = (n, P^P, \sigma)$ and an LTL property ϕ, determining if $\mathcal{A} \models \phi$ is decidable.*

Note that Theorem 8 does not solve our problem, because not every Büchi conversation protocol is realizable. Consider an e-service prototype with four peers, p_a, p_b, p_c, p_d, where $\Sigma_a^{\text{out}} = \Sigma_b^{\text{in}} = \{\alpha\}$, $\Sigma_c^{\text{out}} = \Sigma_d^{\text{in}} = \{\beta\}$, and $\Sigma_a^{\text{in}} = \Sigma_b^{\text{out}} = \Sigma_c^{\text{in}} = \Sigma_d^{\text{out}} = \emptyset$. 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 peers p_a and p_c, so there is no way for them to make sure that α is sent before any β is sent.

4.1 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 $\mathcal{L}^*_{\preceq}(\mathcal{A})$ includes all finite prefix of $\mathcal{L}(\mathcal{A})$ for a Büchi automaton $\mathcal{A}$, clearly $\mathcal{L}^*_{\preceq}(\mathcal{A})$ is regular. Given a conversation protocol $\mathcal{A}$, and a peer (prototype) p_i, the *projection* of $\mathcal{A}$ onto p_i is a Büchi automaton $\mathcal{A}_i$ obtained from $\mathcal{A}$ by replacing each move for a message not in the alphabet of p_i by an ϵ-move. We define $S^P(\mathcal{A})$ to be the e-service derived from the conversation protocol $\mathcal{A}$ based on the e-service prototype S^P where each peer in $S^P(\mathcal{A})$ is $\mathcal{A}_i$. Clearly $\mathcal{L}(\mathcal{A}_i) = \pi_i(\mathcal{L}(\mathcal{A}))$, where π_i is the projection operator w.r.t. peer p_i.

Lossless join property We now introduce the "lossless join" property. Intuitively, the lossless join property requires that a protocol is complete w.r.t. the product of its projection to each peer. Let $S^P = (n, \{p_1, \ldots, p_n\}, \delta)$ be an e-service prototype, the JOIN operator is defined as: $\text{JOIN}(L_1, \ldots, L_n) = \{w \mid w \in \Sigma^{\leq\omega}, \forall i \in [1, n] : \pi_i(w) \in L_i\}$

Definition 9. *Let $\mathcal{A}$ be a conversation protocol for an e-service prototype $S^P = (n, P^P, \delta)$. $\mathcal{A}$ is* lossless join *if* $\mathcal{L}(\mathcal{A}) = \text{JOIN}(\pi_1(\mathcal{L}(\mathcal{A})), \ldots, \pi_n(\mathcal{L}(\mathcal{A})))$.

The check of lossless join property is straightforward. Obtain $S^P(\mathcal{A})$ from $\mathcal{A}$, and then construct the Cartesian product (a generalized Büchi automaton) of $\mathcal{A}_1, \ldots, \mathcal{A}_n$. Then verify whether the resulting product is equivalent to $\mathcal{A}$.

Synchronous compatible property Before we introduce the property, let us revisit the Fresh Market Update example in Fig. 2. We have argued in §3 that the composition in Fig. 2 is bad because the conversation set is not ω-regular, which inhibits the application of model checking techniques. In fact it is even worse. Consider the peer Investor, it is possible that "Data" arrives earlier than "Start", and Investor is blocked. This scenario is similar to the "illegal states" described in [3], where a peer receives an unexpected message. We define an *synchronous compatible* property to avoid such scenarios.

Definition 10. *Let $\mathcal{A}$ be a conversation protocol for an e-service prototype $S^P = (n, P, \delta)$. $\mathcal{A}$ is said to be* synchronous compatible *if for each word $w \in \Sigma^*$ and each message $\alpha \in \Sigma_a^{\text{out}} \cap \Sigma_b^{\text{in}}$, the following holds:*

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

The decision procedure of synchronous compatible property proceeds as follows: construct e-service $S^P(\mathcal{A})$ from $\mathcal{A}$. For every peer 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, i.e., there is a peer ready to send a message while the receiver is not ready to receive. $\mathcal{A}$ is not synchronous compatible if an illegal state is found.

Autonomous property Synchronous compatible property is still not enough to constrain a conversation protocol. Consider the conversation protocol $\mathcal{A}_0$ shown in Fig. 3. It is easy to infer that $\mathcal{A}_0$ is synchronous compatible, however the word $\beta\alpha\gamma\alpha^\omega$ is a legal conversation that is not contained in $\mathcal{L}(\mathcal{A}_0)$.

Taking a close look at the execution paths of all peers (denoted by dotted arrows in Fig. 3), we learn that the abnormal conversation is the result of "ambiguous" understanding of the protocol by different peers, and the racing between A and B at the initial state is the main cause. Consequently, we introduce the following *autonomous* property to restrict racing conditions, so that at any point each peer can make independent decisions to receive, to send or to terminate.

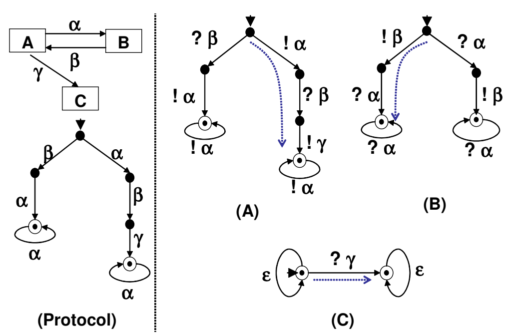

Fig. 3. Ambiguous execution

Let $\mathcal{A}$ be a conversation protocol on a e-service prototype $S^P = (n, P^P, \delta)$. For a peer $p_i \in P^P$, 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 (resp. 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 11. *Let $\mathcal{A}$ be a conversation protocol on e-service prototype $S^P = (n, P^P, \delta)$. $\mathcal{A}$ is* autonomous *if for each peer prototype $p_i \in P^P$ and for each finite prefix $w \in \mathcal{L}^*_{\preceq}(\mathcal{A})$, p_i at $\pi_i(w)$ is only one of the following: output-ready, input-ready, or terminate-ready.*

Given a conversation protocol $\mathcal{A}$ and its e-service prototype $S^P = (n, P^P, \delta)$, we can check the autonomous property as follows. For each peer p_i, let $\mathcal{A}_i = (\Sigma_i^{\text{in}}, \Sigma_i^{\text{out}}, T_i, s_i, F_i)$ be its peer implementation in $S^P(\mathcal{A})$, and let $T_i' \subseteq T_i$ includes each state s where an ϵ loop starting at s passes through at least one final state. Construct prefix automaton $\mathcal{A}_i^*$ for each $\mathcal{A}_i$ by making each state in $\mathcal{A}_i$ a final state. Determinize $\mathcal{A}_i^*$, and now each state of $\mathcal{A}_i^*$ can be represented by a subset of T_i. We check each state s' of $\mathcal{A}_i^*$. When $s' \cap T_i'$ is not empty, we require that there is no outgoing transitions starting from s'. If $s' \cap T_i'$ is empty, then the outgoing transitions from s' are required to be either all output messages or all input messages. The complexity of the above check is NPTIME because of the determinization procedure. The following lemma summarizes the complexity of checking three realizability conditions.

Lemma 12. *Given a conversation protocol $\mathcal{A}$ and an e-service prototype S^P, it can be determined in polynomial time in the size of $\mathcal{A}$ and S^P if $\mathcal{A}$ has lossless join, synchronous compatible, and autonomous property.*

We now proceed to present the main result (Theorem 14), which shows that if the realizability conditions are satisfied, a conversation protocol is realizable.

Lemma 13. *Let $\mathcal{A}$ be a synchronous compatible and autonomous conversation protocol, and $S^P(\mathcal{A})$ the e-service obtained from the projection of $\mathcal{A}$ to each peer, then for each conversation $w \in \mathcal{C}(S^P(\mathcal{A}))$, the following two statements hold*

1. *for each peer p_i, $\pi_i(w) \in \pi_i(\mathcal{L}(\mathcal{A}))$, and*
2. *during any complete run of w, each message sent is consumed eagerly by its receiver, i.e., a peer never sends a message with its queue not empty.*

Proof. (Sketch) We need only prove statement (2) because (1) is implied by (2). Assume $\mathcal{A}$ is synchronous compatible and autonomous, but there is a a run R where a peer p_x sends out a message α_n while a message α_m is stored in its queue. Without loss of generality, let α_m be the first such message during R. Hence each α_i where $i < m$ is consumed eagerly by its receiver. It is not hard to show that for each peer p_i, $\pi_i(\alpha_0 \dots \alpha_{m-1}) \in \mathcal{L}(\mathcal{A}_i)$, now by synchronous compatible definition, $\pi_x(\alpha_1 \dots \alpha_m) \in \mathcal{L}(\mathcal{A}_x)$, and hence p_x is input ready at $\pi_x(\alpha_1 \dots \alpha_{m-1})$. On the other hand, we can infer from the run R that p_x is output ready at $\pi_x(\alpha_1 \dots \alpha_{m-1})$ because of message α_n. This contradicts with the autonomous property, and thus the assumption does not hold.

Theorem 14. *A conversation protocol $\mathcal{A}$ for an e-service prototype S^P is realizable, i.e., $\mathcal{C}(S^P(\mathcal{A})) = \mathcal{L}(\mathcal{A})$, if $\mathcal{A}$ is lossless join, synchronous compatible, and autonomous.*

Following Lemma 12 and Theorem 14, we get the following verification strategy: (1) A conversation protocol is specified by a realizable Büchi automaton, (2) The properties of the protocol are verified on the Büchi automaton specification, (3) The peer implementations for the conversation protocol are synthesized from the Büchi automaton via projection.

The three realizability conditions in Theorem 14 may seem restrictive, however they are satisfied by many real life e-service applications. We verified that five out of the six examples listed on IBM Conversation Support site [19] satisfy the conditions. In fact, except restricting the racing between send and receive actions, our realizability conditions still allow a certain level of parallelism, which makes it acceptable to many e-services.

The results in this section can be directly applied to the framework in [9], and even better results can be achieved. For an FSA conversation protocol $\mathcal{A}$ and its e-service prototype S^P, we can determinize each peer implementation in $S^P(\mathcal{A})$, and it is guaranteed that their composition is deadlock free and peers do not get blocked by unexpected messages in the composition. The difference between the two frameworks is that nondeterministic Büchi automata cannot be determinized.

5 Conclusions

In this paper we studied the global behavior of reactive e-services in terms of the conversations permitted by e-services. LTL model checking on 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 gave three conditions on conversation protocols which ensure realizability. Studying the global behavior of e-services is a promising research topic. The results in [9] and in this paper provide a starting point.

Acknowledgments: Bultan was supported in part by NSF grant CCR-9970976 and NSF Career award CCR-9984822; Fu was partially supported by NSF grant IIS-0101134 and NSF Career award CCR-9984822; Su was also supported in part by NSF grants IIS-0101134 and IIS-9817432.

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. A. Abdulla and B. Jonsson. Verifying programs with unreliable channels. *Journal of Information and Computation*, 127:91–101, 1996.
3. L. D. Alfaro and T. A. Henzinger. Interface automata. In *Proc. of 9th ACM 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*, 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 (BPEL4WS), version 1.1. *available at* http://www.ibm.com/developerworks/library/ws-bpel.

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. of 12th Intl. World Wide Web Conf.*, May 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, 1999.
12. 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.
13. A. Finkel and P. McKenzie. Verifying identical communicating processes is undecidable. *Theoretical Computer Science*, 174(1–2):217–230, 1997.
14. S. J. Garland and N. Lynch. Using I/O automata for developing distributed systems. In *Foundations of Component-Based Systems*. Cambridge Univ. Press, 2000.
15. P. Graunke, R. B. Findler, S. Krishnamurthi, and M. Felleisen. Modeling web interactions. In *Proc. of 12th European Symp. on Programming*, LNCS 2618, 2003.
16. 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.
17. C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.
18. R. Hull, M. Benedikt, C. Christophides, and J. Su. E-services: A look behind the curtain. In *Proc. of 22nd ACM Symp. on Principles of Database Systems*, 2003.
19. IBM. Conversation support project. http://www.research.ibm.com/convsupport/.
20. G. Kahn. The semantics of a simple language for parallel programming. In *Proc. of IFIP 74*, pages 471 – 475. North-Holland, 1974.
21. H. Liu and R. E. Miller. Generalized fair reachability analysis for cyclic protocols. In *IEEE/ACM Transactions on Networking*, pages 192–204, 1996.
22. N. Lynch and M. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proc. 6th ACM Symp. Principles of Distributed Computing*, pages 137–151, 1987.
23. R. Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999.
24. 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.
25. 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.
26. S. K. Rajamani and J. Rehof. A behavioral module system for the pi-calculus. In *Proc. of Static Analysis Symposium (SAS)*, July 2001.
27. Sun. Java message service. http://java.sun.com/products/jms/.
28. W3C. Web service choreography interface (WSCI) version 1.0. *available at* http://www.w3.org/2003/01/wscwg-charter.
29. W3C. Web services description language (WSDL) version 1.1. *available at* http://www.w3.org/TR/wsdl, 2001.