Conversation Specification: A New Approach to Design and Analysis of E-Service Composition

Tevfik Bultan Xiang Fu
University of California
Santa Barbara, CA 93106
USA
{bultan,fuxiang}@cs.ucsb.edu

Richard Hull
Bell Labs, Lucent
600 Mountain Ave. Murray Hill, NJ 07974, USA
hull@research.bell-labs.com

Jianwen Su
University of California
Santa Barbara, CA 93106
USA
su@cs.ucsb.edu

ABSTRACT

This paper introduces a framework for modeling and specifying the global behavior of e-service compositions. Under this framework, peers (individual e-services) communicate through asynchronous messages and each peer maintains a queue for incoming messages. A global "watcher" keeps track of messages as they occur. We propose and study a central notion of a "conversation", which is a sequence of (classes of) messages observed by the watcher. We consider the case where the peers are represented by Mealy machines (finite state machines with input and output). The sets of conversations exhibit unexpected behaviors. For example, there exists a composite e-service based on Mealy peers whose set of conversations is not context free (and not regular). (The set of conversations is always context sensitive.) One cause for this is the queuing of messages; we introduce an operator "prepone" that simulates queue delays from a global perspective and show that the set of conversations of each Mealy e-service is closed under prepone. We illustrate that the global prepone fails to completely capture the queue delay effects and refine prepone to a "local" version on conversations seen by individual peers. On the other hand, Mealy implementations of a composite e-service will always generate conversations whose "projections" are consistent with individual e-services. We use projection-join to reflect such situations. However, there are still Mealy peers whose set of conversations is not the local prepone and projection-join closure of any regular language. Therefore, we propose conversation specifications as a formalism to define the conversations allowed by an e-service composition. We give two technical results concerning the interplay between the local behaviors of Mealy peers and the global behaviors of their compositions. One result shows that for each regular language L, its local prepone and projection-join closure corresponds to the set of conversations by some Mealy peers effectively constructed from L. The second result gives a condition on the shape of a composition which guarantees that the set of conversations that can be realized is the local prepone and projection-join closure of a regular language.

1. INTRODUCTION

The use of e-services (i.e., self-contained Web accessible programs and devices) will revolutionize the way that many e-commerce, consumer software, and telecommunications applications are provided. Emerging standards (e.g., SOAP, UDDI, WSDL, WSFL, BPEL4WS) and industrial technology (e.g., IBM's Web services Toolkit, Sun's Open Net Environment and JiniTM Network technology, Microsoft's .Net and Novell's One Net initiatives, HP's e-speak) in e-services has focused on providing pragmatic, working systems so that e-services can effectively interact with each other. Variants of the web services paradigm also arise in the "converged" network, i.e., the evolving integration of the telephony network and the Internet through standards such as SIP, Parlay/OSA, and 3GPP. Research papers in the field (e.g., [9, 13, 15, 16, 26, 17, 4, 3, 18]) are providing complimentary technologies, for modeling at a more fundamental level both e-services themselves, and frameworks for combining them. The programming language community is addressing the web services phenomenon with new languages [7, 19] and specialized type systems [20]. Recent work on e-services in the semantic web community (e.g., [24, 8, 25, 14]) is beginning to combine tools for annotating e-services and for planning, so that e-services can be combined automatically to achieve a specified functionality.

This work is based on three fundamental observations:

1. There has been essentially no formal work to understand the relationship between the *global* properties of a composite e-service and the *local* properties of the atomic e-services that comprise the composition. Furthermore, our preliminary results reported in this paper indicate that there are unexpected interactions between the local and global behavior of composite e-services. For example, as detailed below if the atomic e-services are described using finite state automata the resulting global behavior cannot always be described in terms of regular languages, although sufficient conditions can be identified to guarantee this state of affairs.

2. Design of composite e-services should incorporate both global and local properties of composite e-services. The traditional *bottom-up* approach to designing these services can lead to undesirable global behaviors. Our initial technical results suggest that an alternative, *top-down* approach to composite e-service design can provide conceptually cleaner services that will be easier to verify and maintain. A formal understanding of the interaction of local and global behaviors of composite e-services will provide an important foundation for the creation of such design and analysis tools.

3. A primary goal of the e-services paradigm is to support the *dynamic* discovery, selection, and composition of (atomic or composite) e-services. Further, a key motivator for this paradigm is the promise of supporting a high degree of *customization* (and personalization) in the provision of services, e.g., through the use of intricate user profile and preferences data, and the use of policy engines in the atomic e-services. Thus, design and analysis tools for composite e-services should be applicable to

WWW2003, May 20–24, 2003, Budapest, Hungary.
ACM 1-58113-680-3/03/0005.xxx.

both dynamic composition of e-services that incorporate policy management for customization.

This paper introduces a framework for modeling and specifying the global behavior of e-service compositions. Under this framework, peers (individual e-services) communicate through asynchronous messages and each peer has a queue for incoming messages. A global "watcher" keeps track of messages as they occur. We propose and study a central notion of a "conversation", which is a sequence of messages observed by the watcher. By understanding properties of these conversations, this study can provide a new approach for the design and analysis of "well-formed" e-service compositions.

Within this general framework, this paper focuses on classes of messages, e.g., in an e-commerce application, the message classes might include, "invoice", "receipt", "acknowledgment", etc. We study the case where the peers are represented by Mealy machines (finite state machines with input and output). The sets of conversations exhibit unexpected behaviors. For example, there exists a composite e-service based on Mealy peers whose set of conversations is not regular nor context free. The set of conversations is shown to be context sensitive. One cause for this is the queuing of messages; we first introduce an operator "prepone" in an attempt to simulate queue delays. Although the set of conversations of each composite e-service with Mealy peers is closed under prepone, we illustrate that prepone does not completely capture the queue delay effects. We refine the prepone operator to a "local" version which applies to conversations by individual peers. Another aspect of the composite e-service is that decisions are only made by individual e-services possibly with communications with each other. This means that each e-service sees only a "local" view of the global conversation. Consequently, Mealy implementations in the composite e-service will also include conversations that whose "projections" to individual e-services are consistent with the local e-services. We use projection-join closure to capture such situations. This is reminiscent of the decomposition and join in the relational databases. However, there are still Mealy peers whose set of conversations is not prepone and projection-join closure of any regular language. Therefore we propose conversation specifications as a formalism to define the conversations allowed by an e-service composition.

In this paper we present two technical results concerning the interplay between the local behaviors of Mealy peers and the global behaviors of their compositions. One result shows that for each regular language L its local prepone and projection-join closure corresponds to the set of conversations by some Mealy peers effectively constructed from L. The second result gives a condition on the shape of a composition which guarantees that the set of conversations that can be realized is the local prepone and projection-join closure of a regular language.

The paper is organized as follows. Section 2 presents a formal framework for studying composite e-services. Sections 3, 4, and 5 present some preliminary results that focus on an abstract view of the formal framework based on the classes of messages passed between e-services and finite state automata; results here illustrate the unexpected nature of the interplay between local and global in composite e-services. Section 6 concludes the paper.

2. A MODEL FOR E-SERVICES

In this section we describe a paradigm for modeling e-services and discuss various modeling issues within the paradigm. Our goal is to set up the ground work for studying composition of e-services. For this purpose, we start with a very general abstract model for e-services in this section, and gradually bring in some refinements that are relevant to our investigation in the later sections.

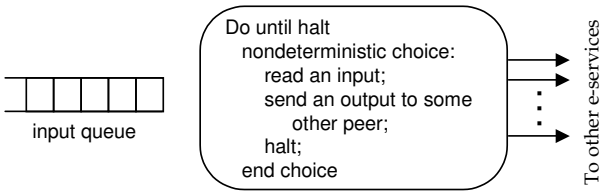

Figure 1: A model of e-service

While abstract and focused primarily on global behavior, our paradigm is based on the fundamental constructs of the web services as promoted by, e.g., BPEL4WS [11], the SOAP standard, IBM's Web services Toolkit, Microsoft's .Net, and other industrial products and proposals. It also follows the model adopted by much of the research on web service composition [8, 25], work on web service programming languages [7, 19], and the AZTEC prototype [17]. Importantly, our model also reflects fundamental constructs emerging for the next generation telecommunications network. The telecommunications network has traditionally involved a small number of monolithic, multi-function switches, but is now migrating to an Internet style that is "disaggregated" with a high number of distributed, specialized softswitches and feature servers. Indeed, the Session Initiation Protocol (SIP) [27] provides a highly flexible mechanism for coordination of low-level telecomm services that is reminiscent of, but less expressive than, SOAP. Also, the reference architecture for an IP Multimedia Core Network Subsystem (IMS) that is being proposed by the 3GPP [1] and 3GPP2 [2] standards bodies (for 3G wireless data and voice) is quite compatible with the core elements of the web services paradigm.

A fundamental observation is that an e-service (1) provides services through "service sessions" and (2) reacts to "events" during a session, although the implementation of an e-service may be very complex. Fig. 1 illustrates an abstraction of an *e-service* (called here a *peer*) as a program that processes the input events from an input queue and determines the response if any (in the form of outgoing events) and termination. For the present we make no assumptions about the computational power of a peer, nor how much storage it has, etc.

Events form the enabling mechanism in composing e-services. In this paper, we focus on an important kind of event—"messages" between the individual e-services. Messages are organized into a finite collection of "message classes" (each message is in exactly one class). Message classes can be used to simplify and organize the specification of actions. A *message class* consists of a name and a finite set of attributes. A *message* of a class m consists of an identifier of an e-service enactment (session), an identifier for the message itself, the sender, the receiver, and a function mapping each attribute of m to a value (of appropriate type).

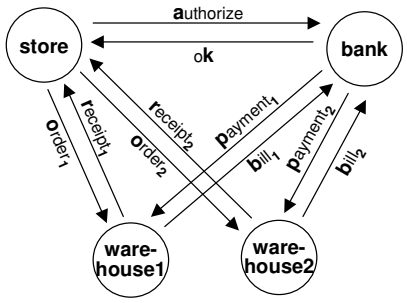

Figure 2: A composite e-service

EXAMPLE 2.1. Consider a very simple example of e-services involving four servers: a retail *store* that plans to replenish its inventory, its *bank*, and two *warehouses* that supply goods. Fig. 2 shows the four servers and message classes between them. In a (simplified) typical scenario, the store requests an authorization from the bank; after receiving the approval from the bank, the store can send one or more orders to the warehouses. When a warehouse receives an order, it responds by billing the bank for the amount on the order, and sends the store a receipt. The bank, in turn, makes a payment after receiving a bill. The message class authorize may include attributes "date", "requested amount", "account number", etc. and an authorize message may look like:

$\langle 123, p12,$ store, bank, "11-1-2002", "$2,500", 43-56483, ...$\rangle$. ∎

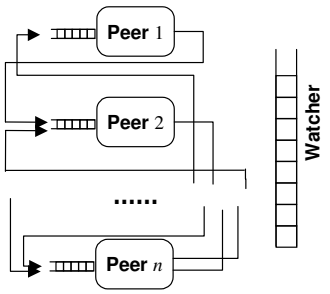

Figure 3: E-service composition

Unsurprisingly, the behavior of each server in Example 2.1 corresponds to the abstract model (Fig. 1). The composed e-service can also be represented using this abstract model. Fig. 3 shows an abstract architecture for an e-service composition, where the output of an e-service goes to the input queue of another e-service. The *watcher* is the concatenation of all the messages exchanged among the peers. One can think of the watcher as a person listening to the network and recording, one by one, each message that has been sent over the network. One of the central insights in this paper is that postulating the (conceptual) existence of the watcher permits two complimentary perspectives on composite e-services design and analysis, namely, *top-down* vs. *bottom-up*. In particular, specifying the desired global behavior as observed by the watcher is fundamentally different from specifying the behaviors of the peers which generate that watcher behavior.

Roughly, a peer implementation can be viewed as a "program" that decides, based on the received messages and the messages already sent, if a new message should be sent, and/or if the session should terminate. AZTEC [17] classifies e-services into two types: (1) *Discrete e-services* that do not allow interactions during the service, and (2) *Interactive e-services* that allow arbitrarily many interactions during the service, such as VCR type of controls during a session. A discrete e-service can be viewed as a service whose output depends only on the original input, while in an interactive e-service inputs can be unpredictable and the e-service reacts to input as they occur. AZTEC [17] emphasizes the importance of interactive e-services in the context of telecommunications applications, but they are also relevant in the context of e-commerce. A single occurrence of ordering a book can be modeled using discrete e-services. But in many cases a vendor wants to track the entire relationship with a customer, and perhaps modify customer treatment accordingly (e.g., frequent flier programs). In this case, some of the e-services used are fundamentally long-running and interactive.

We now start to lay the foundation for a formal study of global behavior of composite e-services. The first step is to formalize the notion of a "schema" for a composite e-service.

DEFINITION. An *e-composition schema* (*ec-schema*) is a triple (M, P, C), where M is a finite set of message classes, P a finite set of (abstract) peers (e-services), and C is a finite set of one-way communication channels, [1] i.e., C is a finite set of triples (p_s, p_r, σ) such that $p_s, p_r \in P$, $p_s \neq p_r$, $\sigma \subseteq M$, and for each pair of channels (p_s, p_r, σ) and (p'_s, p'_r, σ') in C, if $p_s = p'_s$ and $p_r = p'_r$, then $\sigma = \sigma'$; otherwise, $\sigma \cap \sigma' = \varnothing$.

Let $S = (M, P, C)$ be an ec-schema. In a channel $(p_s, p_r, \sigma) \in C$, p_s is the *sender* and p_r the *receiver* of the channel, and only messages in classes in σ are allowed to be sent on the channel. We fix the following notations in the remainder of the paper. For each peer $p \in P$, Σ_p^{in}, Σ_p^{out} denote the sets of (classes whose) messages may be put in the input queue of p and in the output by p, resp. i.e., $\Sigma_p^{\text{in}} = \cup\{\sigma \mid (p_s, p, \sigma) \in C\}$ and $\Sigma_p^{\text{out}} = \cup\{\sigma \mid (p, p_r, \sigma) \in C\}$. Finally, let $\Sigma = \bigcup_{p \in P} \Sigma_p^{\text{in}} = \bigcup_{p \in P} \Sigma_p^{\text{out}} \subseteq M$.

DEFINITION. Let $S = (M, P, C)$ be an ec-schema and $p \in P$. A *(peer) implementation* of p is a computable function which maps a sequence of incoming and outgoing messages (a word over Σ) to $\Sigma \cup \{\text{halt}, \text{no-op}\}$. An *ec-implementation* of S is a mapping I such that for each $p \in P$, $I(p)$ is an implementation of p.

The e-service implementation described above is very general. In the following sections, we introduce a specific type of implementations based on Mealy machines and study the global behavior of a composition and the local behaviors of individual e-services.

3. MEALY IMPLEMENTATIONS

A primary concern in composing multiple e-services is to specify global behavior of the composition by limiting the way the e-services are to interact with each other, e.g., the coordination of messages. In order to understand the global behavior, we focus on the families of sequences of messages among the peers.

In the technical discussions, we consider a special family of implementations called "Mealy implementations" (or "Mealy peers") based on Mealy machines [22]. There are two primary reasons. First, Mealy machines are a variant of finite state machines and seem suitable for modeling e-services. Second, Mealy implementations make it possible to analyze some aspects of the global behavior. As we shall see, our preliminary results suggest a "top-down" approach to e-service compositions and raise many interesting questions.

Let ϵ denote the empty string. If Γ is an alphabet, we define $\widehat{\Gamma} = \Gamma \cup \{\epsilon\}$ (the extended alphabet with the empty string).

DEFINITION. Let $S = (M, P, C)$ be an ec-schema and $p \in P$. A *Mealy implementation* of p is a (nondeterministic) Mealy machine $(T, s, \Sigma_p^{\text{in}}, \Sigma_p^{\text{out}}, F, \delta)$ where T is a finite set of states, $s \in T$ the starting state, $F \subseteq T$ a set of final states, Σ_p^{in} and Σ_p^{out} are derived from S as before, and $\delta : T \times \widehat{\Sigma}_p^{\text{in}} \to 2^{T \times \widehat{\Sigma}_p^{\text{out}}}$ is a transition function such that it either consumes a nonempty input or produces a nonempty output but not both (empty moves are allowed). An ec-implementation is *Mealy* if its peer implementations are all Mealy.

A Mealy implementation of a peer reacts to messages according to their classes while ignoring the contents. Although Mealy implementations are finite state machines, they can model e-services in many applications nicely. This is illustrated by Example 2.1, where

[1] We use the term channel to identify the sender and the receiver of a message, not how it is exchanged. In our model, messages are exchanged through a common medium that is shared by all peers and the watcher records the messages exchanged through this common medium.

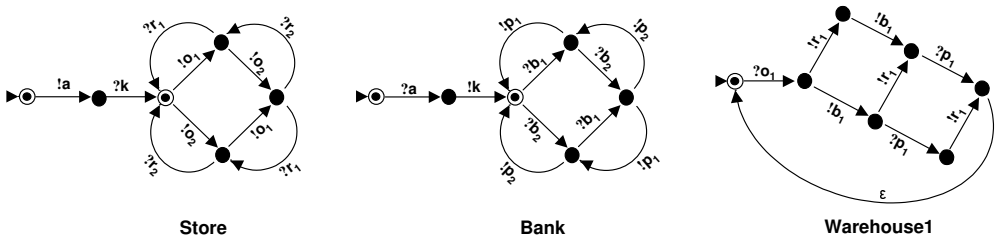

Figure 4: A Mealy implementation for the Warehouse example

the message classes effectively dictate the actions to be taken by each server and consequently the responses.

EXAMPLE 3.1. Fig. 4 shows a family of Mealy peer implementations for the warehouse example of Example 2.1. (The implementation for Warehouse2 is analogous to the implementation for Warehouse1.) In these diagrams, we use "$!a$" ("$?a$") to denote sending (receiving) a message from class a. It can be verified that the ec-language generated by this Mealy implementation

$$ak \,\mathrm{SH}((o_1 \,\mathrm{SH}(r_1, b_1p_1))^*, (o_2 \,\mathrm{SH}(r_2, b_2p_2))^*),$$

where SH is the shuffle operator. Thus, this implementation corresponds to the case where the timing of sending **r**eceipt$_1$ from Warehouse1 to Store is independent of the timing of the corresponding messages **b**ill$_1$ and **p**ayment$_1$ between Bank and Warehouse1. By using a different implementation for Warehouse1 a specific sequencing could be enforced, e.g., $o_1b_1p_1r_1$. ∎

We now define the notion to capture the computation of ec-implementations.

Let $S = (M, P, C)$ be an ec-schema where $P = \{p_1, ..., p_n\}$. Suppose that I is a Mealy ec-implementation for S. An *ec-configuration* of I is a $(2n + 1)$-tuple of the form

$$(Q_1, t_1, ..., Q_n, t_n, w)$$

where for each $1 \leqslant j \leqslant n$, $Q_j \in (\Sigma^{\mathrm{in}}_{p_j})^*$, $t_j \in T_j$ (i.e., states of $I(p_j)$), and $w \in \Sigma^*$.

For ec-configurations $\gamma = (Q_1, t_1, ..., Q_n, t_n, w)$ and $\gamma' = (Q'_1, t'_1, ..., Q'_n, t'_n, w')$, we say that $\gamma \to \gamma'$ if one of the following three conditions holds:

- (Peer p_j executes an ϵ-move) there exists $1 \leqslant j \leqslant n$ such that
  1. $(t'_j, \epsilon) \in \delta_j(t_j, \epsilon)$,
  2. $Q'_j = Q_j$,
  3. for each $k \neq j$, $Q'_k = Q_k$ and $t'_k = t_k$, and
  4. $w' = w$.
- (Peer p_j consumes an input) there exist $1 \leqslant j \leqslant n$ and $\alpha \in \Sigma^{\mathrm{in}}_{p_j}$ such that
  1. $(t'_j, \epsilon) \in \delta_j(t_j, \alpha)$,
  2. $Q_j = \alpha Q'_j$,
  3. $Q_k = Q'_k$ for each $k \neq j$,
  4. $t'_k = t_k$ for each $k \neq j$, and
  5. $w' = w$.
- (Peer p_j sends an output to peer p_k and writes to the watcher) there exist $1 \leqslant j, k \leqslant n$ and $\beta \in \Sigma^{\mathrm{out}}_{p_j} \cap \Sigma^{\mathrm{in}}_{p_k}$ such that
  1. $(t'_j, \beta) \in \delta_j(t_j, \epsilon)$,
  2. $Q'_k = Q_k\beta$,
  3. $Q'_l = Q_l$ for each $l \neq k$,
  4. $t'_l = t_l$ for each $l \neq j$, and
  5. $w' = w\beta$.

we denote by $\xrightarrow{*}$ the reflexive and transitive closure of $\to$.

DEFINITION. Let $S = (M, P, C)$ be an ec-schema where $P = \{p_1, ..., p_n\}$ and I a Mealy ec-implementation of S. A word w over Σ is a (*halting*) *conversation* for I if

$$(\epsilon, s_1, ..., \epsilon, s_n, \epsilon) \xrightarrow{*} (\epsilon, t_1, ..., \epsilon, t_n, w)$$

where for each $1 \leqslant j \leqslant n$, s_j is the starting state and t_j a (final) state in the Mealy machine $I(p_j)$. We call the above sequence of ec-configurations an *ec-run* of w. The *ec-language* of I, $\mathcal{C}(I)$, is the set of all halting conversations for I.

While Mealy peers resemble I/O automata [23] and interface automata [6, 5], the communication model is different. In our composition model, Mealy peers communicate asynchronously. Specifically, a queue is used for each peer to buffer messages that were received but not processed so far. In approaches such as CSP [21], and I/O and interface automata, the communicating processes execute a send and a corresponding receive action synchronously. This makes Mealy implementations significantly different from the communication model used in approaches such as CSP, I/O and interface automata. Our model of Mealy peers is similar to Communicating Finite State Machines defined in [12]. However, in our model messages are exchanged through a common medium and then stored in the queues of the peers, whereas in [12] each pair of communicating machines use isolated communication channels. Our goal is to investigate the global behavior of the protocol by investigating the possible configurations of the watcher which models the behavior of this common medium. Finally, [10] studies "quasi-realtime" automata with queues. These are single automata with one or more queues, where an automaton can write a bounded number of letters on the queue(s) for each input letter read. In [10] the input and queue alphabets maybe different; in our framework the alphabets are identical.

4. CONVERSATIONS BY MEALY PEERS

This section presents examples illustrating unexpected behavior of Mealy implementations of ec-schemas. These motivate the identification of two key closure properties of Mealy ec-languages, and lead to some characterizations of Mealy ec-languages.

EXAMPLE 4.1. Figure 5 shows a Mealy implementation I_{ab} with two peers. Peer p_1 sends requests a while p_2 responds with a b message for each a message. Since a messages can be temporarily stored in the queue of p_2, the ec-language $\mathcal{C}(I_{ab})$ consists of words with the same number of a's as b's and each b has a corresponding a that occurs somewhere beforehand. Note that $\mathcal{C}(I_{ab}) \cap (a^*b^*) = \{a^nb^n \mid n \geqslant 0\}$. Therefore $\mathcal{C}(I_{ab})$ is not regular (but it is context free). ∎

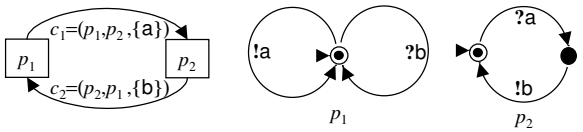

Figure 5: A Mealy implementation for Example 4.1

Interestingly, if a Mealy ec-implementation I is restricted to the synchronous communication mode (i.e., a send and the corresponding receive are done at the same time and queues are basically empty), it is easy to observe that $\mathcal{C}(I)$ is always regular.

Using an idea similar to that in Example 4.1, one can easily construct a Mealy ec-implementation whose ec-language is not regular nor context free but context sensitive. However,

THEOREM 4.2. *Let I be an arbitrary Mealy ec-implementation of any ec-schema.*

(a) $\mathcal{C}(I)$ is context sensitive.

(b) There exists a finite state, quasi-realtime automaton M with 3 queues that accepts $\mathcal{C}(I)$.

(c) If the computations of conversations are restricted to only allow queues with length bounded by a fixed constant, then the restricted ec-language $\mathcal{C}^{\text{bounded}}(I)$ is regular.

Theorem 4.2 highlights differences between the synchronous communication models in I/O and interface automata and the asynchronous model described here. The main part of the proof for (a) is to observe that $\mathcal{C}(I)$ can be recognized by a linear bounded automaton. Part (b) follows from a result of [10], stating that each quasi-realtime automaton with n queues can be simulated by one with 3 queues. Part (c) can be proved by a relatively straightforward argument based on the closure of regular languages under intersection.

We now return to the phenomenon exposed by Example 4.1. A close examination indicates that the primary reason for this behavior is that the message queue of a peer serves as a "buffer" for the input: while conversations monitor the arrival of messages at the queues the messages may not be read right away. To understand this effect, we introduce the operator PREPONE on the alphabet Σ of an ec-schema as follows.

Let $w = w'm_1m_2w''$ be a word in Σ^*, where m_1 is in the set of messages on the channel from p_1 to p_1' and m_2 in the set of messages on the channel from p_2 to p_2'. If either (1) $\{p_1, p_1'\}$ and $\{p_2, p_2'\}$ are disjoint, or (2) $p_1 = p_2'$ and $p_1' \neq p_2$, then PREPONE(w) includes the word $w'm_2m_1w''$. Intuitively, the operator PREPONE allows two messages in a conversation to be swapped if the senders and receivers are completely disjoint, or a later message to a peer can arrive in the queue earlier than an outgoing message from the peer since the outgoing message cannot depend on a later arrived message.

It is important to note that PREPONE applies to the global sequence of messages observed by the watcher. We will exhibit later in the section that PREPONE is not strong enough to characterize the behaviors of Mealy ec-implementations.

If L is a language over Σ, we define PREPONE(L) to be the smallest language that contains L and is closed under PREPONE. The following interesting property holds for PREPONE.

LEMMA 4.3. *For each Mealy ec-implementation I of an ec-schema,* PREPONE$(\mathcal{C}(I)) \subseteq \mathcal{C}(I)$ *(closure under* PREPONE*).*

Since the set of context-sensitive languages does not have the PREPONE closure property, the following holds.

COROLLARY 4.4. *There is a context-sensitive language L such that $L \neq \mathcal{C}(I)$ for any Mealy ec-implementation I.*

The second property of ec-languages concerns with combining "local views" of conversations into global conversations, this is reminiscent of the join operator in the relational database model.

EXAMPLE 4.5. Consider an ec-schema that has four peers p_1, p_2, p_3, p_4 and three channels $(p_1, p_2, \{a\})$, $(p_3, p_4, \{b\})$, and $(p_4, p_3, \{c\})$. Is there any Mealy ec-implementation that generates the regular language $\{a, bc\}$? Note that the peer groups $\{p_1, p_2\}$ and $\{p_3, p_4\}$ are in fact independent; there is no communication possible between them. It can be shown that any Mealy ec-implementation that generates $\{a, bc\}$ also generates each of ϵ, abc, bac, and bca. ∎

The above example suggests that if two global behaviors have exactly the same local views, they are indistinguishable. We formalize this concept below.

The definition of $\xrightarrow{*}$ given for Mealy ec-implementations has the effect of *generating* words. We now define a kind of converse for individual Mealy peer implementations, which has the effect of *consuming* words. Let f be a Mealy implementation $(T, s, \Sigma_p^{\text{in}}, \Sigma_p^{\text{out}}, F, \delta)$ for a peer p. Let $\Sigma_p = \Sigma_p^{\text{in}} \cup \Sigma_p^{\text{out}}$. A *local (l-)configuration* of f is is a triple $(t, u, v) \in T \times (\Sigma_p^{\text{in}})^* \times \Sigma_p^*$. In an l-configuration (t, u, v), t is the current state of the peer p, u is the sequence of messages in the input queue of p, v is a sequence of "future messages" including the incoming messages not yet in the queue of p and the messages to be sent out by p (i.e., v represents the remaining portion of a conversation projected to the messages visible to p_i).

We define $(t, u, v) \rightarrow_f (t', u', v')$ for a pair of l-configurations (t, u, v) and (t', u', v') if one of the following holds for some $a \in \Sigma_p^{\text{in}}$ and some $b \in \Sigma_p^{\text{out}}$:

- (Consuming a message from the queue) $u = au'$, $v = v'$, and $(t', \epsilon) \in \delta(t, a)$,
- (Sending a message) $u = u'$, $v = bv'$, and $(t', b) \in \delta(t, \epsilon)$,
- (ϵ-move) $u = u'$, $v = v'$, and $(t', \epsilon) \in \delta(t, \epsilon)$, or
- (Enqueuing a message) $t = t', u' = ua, v = av'$.

We denote by $\xrightarrow{*}_f$ the reflexive and transitive closure of $\rightarrow_f$.

Let $w \in \Sigma_p^*$. A *local (l-)run of* w is a (finite) sequence of l-configurations $c_0 = (s, \epsilon, w), c_1, ..., c_k$ $(k \geqslant 0)$ such that $c_i \rightarrow_f c_{i+1}$ for $0 \leqslant i < k$. A word w in the language Σ_p^* is a *(halting) execution* of f if $(s, \epsilon, w) \xrightarrow{*}_f (q, \epsilon, \epsilon)$ for some (final) state q.

For word $w \in \Sigma^*$ and peer p, let $\pi_p(w)$ denote the restriction of w to the set Σ_p $(= \Sigma_p^{\text{in}} \cup \Sigma_p^{\text{out}})$.

LEMMA 4.6. *Let I be a Mealy ec-implementation of an ec-schema S. Let $w \in \Sigma^*$. If for each peer p, $\pi_p(w)$ is a (halting) execution of p, then w is a (halting) conversation for I. The converse is also true.*

PROOF. (Sketch) Let $w = \alpha_1 \cdots \alpha_m$ be a word over Σ. We outline a proof for the direction "if projection of w to each peer is a (halting) execution, then w is a (halting) conversation". The converse is trivial.

Without loss of generality, we assume that the ec-schema has peers $p_1, ..., p_n$. Since for each peer p_i, the projection $\pi_i(w)$ is a local execution, there exists a corresponding l-run γ_i for $\pi_i(w)$. We show that w is a (halting) conversation by constructing an ec-run that simulates each γ_i. The construction has $(m+1)$ *phases*. Phase

0 is the initialization phase where we simulate in the global ec-run the initial ϵ-moves of each p_i until it advances to an l-configuration that is ready to do a send-message action or an enqueue-message action. Then in each phase j, we simulate the transmission of message α_j, where only the sender and receiver of α_j are involved. We start with the sender of α_j. We execute the send-α_j action, and its follow-up actions such as ϵ-moves and consume-message actions, until we encounter an enqueue-message or send-message action on a message $\alpha_{j'}$ where $j' > j$. Then we turn to the receiver of α_j, execute the enqueue-α_j action and the follow-up actions until an action related to a later message is reached.

We can prove the correctness of the above process by an induction on the number of phases. Specifically, it can be shown that (1) after the completion of phase j, the l-run of each peer p_i has been simulated right before the last l-configuration which contains the future messages $\alpha_{j+1} \cdots \alpha_m$, and (2) the simulation can always proceed. It follows that the last global ec-configuration is consistent with the last l-configuration for every γ_i. ∎

Now we define the join operator $\bowtie$ which takes as input a sequence of languages $L_1, ..., L_n$ where each $L_i \subseteq \Sigma_{p_i}^*$ is a set of words for peer p_i, and n is the number of peers in the ec-schema. It returns a language over Σ.

$$\bowtie_i L_i = \{w \mid \forall i\ \pi_{p_i}(w) \in L_i\}$$

Then Lemma 4.6 implies the following.

LEMMA 4.7. *For each Mealy ec-implementation I of an ec-schema with peers $p_1, ..., p_n$, $\bowtie_i \pi_{p_i}(\mathcal{C}(I)) \subseteq \mathcal{C}(I)$.* ∎

Given a language L over Σ, let $closure(L)$ denote the minimal superset of L that is closed under PREPONE and $\bowtie$. From Lemmas 4.3 and 4.7, we can infer that for each Mealy ec-implementation I, the following holds:

$$L \subseteq \mathcal{C}(I) \Rightarrow closure(L) \subseteq \mathcal{C}(I)$$

Essentially, this states that any Mealy ec-implementation that generates the behavior set L must also generate its closure. One interesting question is given L, is it always possible to synthesize a Mealy ec-implementation I such that $\mathcal{C}(I) = closure(L)$?

The answer unfortunately is negative. Consider the following example.

EXAMPLE 4.8. Shown in Figure 6 is an ec-schema that consists of three peers and three channels. The language $L = \{ab, bac\}$, and it is obvious that $closure(L) = L$.

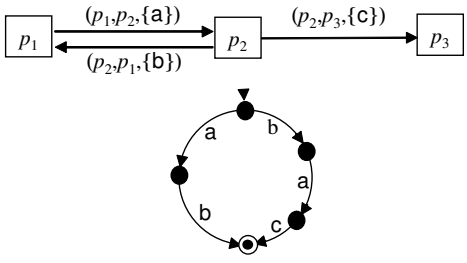

Figure 6: Yet another example

Let I be an arbitrary Mealy ec-implementation satisfying the condition $closure(L) \subseteq \mathcal{C}(I)$. Consider the local run on peer p_2 for the conversation bac. The send of c must be after b, however the consumption of a may be after the send of c. This implies that bac or bca or both must be accepted by $I(p_2)$. Similarly we can infer that ab must be accepted by $I(p_1)$.

If bac is recognized by $I(p_2)$, consider the scenario that p_1 takes the local execution path ab and p_2 takes the path bac. It is not hard to see that abc is a conversation since p_2 sends b while having a in its input queue. Thus $closure(L) \neq \mathcal{C}(I)$. ∎

One reason that the $closure(L)$ cannot be the set of conversations by some Mealy ec-implementation is that PREPONE applying to (global) conversations is too weak. Consider the projection of the conversation abc on p_2 in Example 4.8. If it is not accepted by $I(p_2)$, it must be the result of applying one or more "prepone" like swaps on an accepted word. For example, swap the sequence of output message b and input message a, we get abc from bac. Note that this type of swap differs from PREPONE since the former is applied locally instead of globally. Secondly, we allow the receiver of the first message and the sender of the second message to be the same, which is forbidden in PREPONE. We call this type of swap a *local prepone* defined below. For each peer p_i, if

$$w = w' m_1 m_2 w''$$

is a word in Σ_i^*, where p_i is the sender of m_1 and the receiver of m_2, then word $w' m_2 m_1 w''$ is in $\text{LP}_i(w)$.

Using local prepone operators and $\bowtie$, we define for each language L over Σ the *ec-closure* of L as follows.

$$ecc(L) = \bowtie_i \text{LP}_i^*(\pi_{p_i}(L)),$$

where LP_i^* represents the reflexive and transitive closure of LP_i, for each peer p_i. It is easy to see that $closure(L) \subseteq ecc(L)$. In Section 5 we shall show that $ecc(L)$ can always be synthesized for each regular language L.

Now let us consider the inverse of the synthesis problem. Given a Mealy ec-implementation I, can we find a regular language as its core? The following example provides a negative answer.

EXAMPLE 4.9. Consider an ec-schema shown in Figure 7 consisting of 3 Mealy peers, p_1, p_2, p_3 and 3 channels. Intuitively, p_1 sends, say i messages of class a, to p_2, a message b to p_3, and then halt; p_2 responds to each a message by send a c message to p_3; p_3 expects b at the beginning and then consumes all c messages. It is not hard to see that the only way for p_3 to halt is for p_2 to keep all a messages in its queue till after p_1 sends b to p_3. Thus $L = \{a^i b c^i \mid i \geqslant 0\}$ is its ec-language. It can be shown that each subset L' of L satisfies the following property

$$L' = closure(L') = ecc(L')$$ ∎

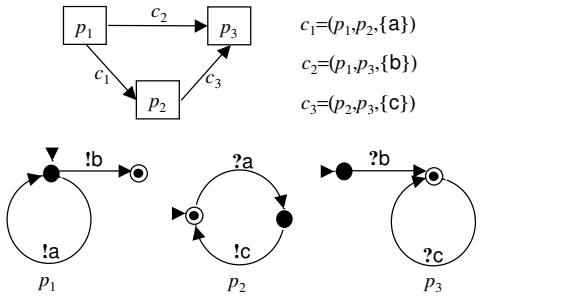

Figure 7: A Mealy implementation for Example 4.9

Example 4.9 suggests the following.

PROPOSITION 4.10. *There exists a Mealy ec-implementation I such that $\mathcal{C}(I) \neq ecc(L)$ for each regular language L.*

5. CONVERSATION SPECIFICATION

Proposition 4.10 suggests that adding asynchronous communication significantly increases the power of essentially finite state machines (Mealy peers). This unsettling fact suggests that focusing only on peers in e-service composition design is fundamentally flawed. Attention has to be given on the "global behavior" of such composed machineries early on. The *conversation* among the peers models the global behavior that we would like to capture. Although one can try to *reason* about the global behavior after specifying (designing) individual peers through the composition, it may be "cheaper" and more direct to provide a specification of the global behavior.

DEFINITION. Let $S = (M, P, C)$ be an ec-schema. A *conversation specification* for S is a specification $\mathcal{S}$ (e.g., by regular expression, finite state automaton, intertask dependencies, etc.) of a language over Σ. The language specified by $\mathcal{S}$ is denoted $L(\mathcal{S})$. Let $\mathcal{S}$ be a conversation specification. An ec-implementation I of S *conforms to* $\mathcal{S}$ if $\mathcal{C}(I)$ is contained in the ec-closure of $L(\mathcal{S})$, and *realizes* $\mathcal{S}$ if $\mathcal{C}(I) =$ the ec-closure of $L(\mathcal{S})$.

We are interested in the following question: For a conversation specification $\mathcal{S}$, can we construct a (Mealy) ec-implementation I that realizes $\mathcal{S}$? The main result of the section is to show that the answer is positive.

THEOREM 5.1. *For every regular language L, one can effectively construct a Mealy ec-implementation I that realizes L.*

We now discuss the proof of the above theorem. The proof consists of the following main steps. First, we construct a finite state automaton G that accepts L. From G, we construct, for each peer p_i, a Mealy implementation G_i. This construction is essentially a projection: replace all edges in G that are irrelevant to channels connected to p_i by ϵ moves, change edges of messages sent to p as input, and finally edges of messages sent by p_i as output. To prove that the composition of the implementation G generates exactly $ecc(L)$, we have to show that

$$\mathcal{C}(I) = \bowtie_i \ \text{LP}_i^*(\pi_{p_i}(L)) \tag{1}$$

By Lemma 4.6, a word w is contained in $\mathcal{C}(I)$ if and only if for each peer p_i, $\pi_{p_i}(w)$ is a halting execution. Combined with the fact that $L(G_i) = \pi_{p_i}(L)$, we can infer that to prove Equation (1), it suffices to show the following property (Lemma 5.2).

LEMMA 5.2. *Let M_i be a Mealy implementation for peer p_i. A word $w \in \Sigma_{p_i}^*$ is a halting execution if and only if $w \in \text{LP}_i^*(L(M_i))$.*

PROOF. (Sketch) We prove by induction that $w \in \text{LP}_i^*(L(M_i))$ is a sufficient condition for w being a local execution. In the induction proof it suffices to show the claim that if a word w is contained in $\text{LP}(w')$ for some local execution w', w is also a local execution. The proof of the claim is straightforward, because we can always construct an l-run for w by modifying the l-run of w'.

Next we show that $w \in \text{LP}_i^*(L(M_i))$ is a necessary condition. We show that for any halting execution w, we can always find $w' \in L(M_i)$ such that $w \in \text{LP}^*(w')$, by applying "reverse prepone" procedure finitely many times. We briefly describe the procedure below. Consider the l-run $c_0 \rightarrow_{M_i} \cdots \rightarrow_{M_i} c_n$ of the local execution w. Let $(q_a, u_1, \alpha u_2) \rightarrow_{M_i} (q_{a+1}, u_1, u_2)$ be the first send-message action such that input queue is not empty, i.e., $|u_1| > 0$. It is not hard to show that w can be written as $w = w_1 u_1 \alpha u_2$, where w_1 includes those eagerly processed messages before the arrival of any message in u_1. Now let $w^1 = w_1 \alpha u_1 u_2$, we can show that $w \in \text{LP}_i^{|u_1|}(w^1)$ and w^1 is also a halting execution. Repeat the above procedure, until we cannot find a send-message action with a non-empty queue, then we get a list $w^0, w^1, ..., w^k$ where $w^0 = w$, $w^k \in L(M_i)$, and for each $0 \leqslant j < k$, $w^j \in \text{LP}_i^*(w^{j+1})$. The last word w^k is the w' we are looking for. ∎

Lemma 5.2 implies the following corollary.

COROLLARY 5.3. *Given a Mealy ec-implementation I for an ec-schema S. The conversation set generated by I is the following:*

$$\mathcal{C}(I) = \bowtie_i \text{LP}_i^*(L(I(p_i)))$$

Corollary 5.3 does not mean that there must be a regular language "core" for a Mealy ec-implementation. We can give a characterization for a subclass which guarantees such a regular core.

DEFINITION. Let $S = (M, P, C)$ be an ec-schema. Let $G(S) = (P, E)$ be the non-directed graph where

$$E = \{\{p, p'\} \mid (p, p') \in C\}.$$

Schema S is *tree-based* if $G(S)$ is a (non-directed) tree.

PROPOSITION 5.4. *Let S be a tree-based ec-schema and I a Mealy implementation for S. Then $\mathcal{C}(I) = ecc(L)$ for some regular language L.*

PROOF. (Sketch) First we extend the notion of conversation to indicate when messages are read from an input queue, in addition to when messages are written onto the input queue. Let $read(\Sigma) = \{\alpha^r \mid \alpha \in \Sigma\}$. A *read-augmented conversation* of implementation I is a word v over the alphabet $\Sigma \cup read(\Sigma)$ that corresponds to a computation over I, where each occurrence of a letter α^r corresponds to a time when letter α was read from a peer's input queue. Given such a v, $\pi_\Sigma(v)$ denotes the projection of v onto the alphabet Σ.

Now let S and I be as in the statement of the proposition. A key lemma is to show that if v is a read-augmented conversation of I corresponding to a halting computation, then there is a read-augmented conversation v' of I such that $\pi_\Sigma(v) \in \text{PREPONE}(\pi_\Sigma(v'))$ and v' has the *immediate read property*, that is, for each occurrence o of $\alpha \in \Sigma$ occurring in v' there is an occurrence of α^r immediately following o in v'. The key idea of the proof is that since S is tree-based, entire blocks of a computation occurring in one "part" of the tree (if partitioned by removing p) can be "delayed" or "accelerated" so that a message is not put onto the queue of peer p until p is ready to read that message.

From the above key lemma, we learn that each halting conversation $w \in \mathcal{C}(I)$ is contained in $\text{PREPONE}(w')$ for some word w', where w' satisfies the following condition: for each peer p_i, the projection $\pi_{p_i}(w')$ is accepted by $I(p_i)$. Let $L = \bowtie_i L(I(p_i))$, it is not hard to show that L is a regular language, and $\mathcal{C}(I) = \text{PREPONE}(L)$. Let I' be the Mealy ec-implementation generated from the projection of L to each peer. It is easy to infer that $\mathcal{C}(I') \subseteq \mathcal{C}(I)$. Combined with the known fact that

$$\text{PREPONE}(L) \subseteq closure(L) \subseteq ecc(L) = \mathcal{C}(I')$$

we can further infer that

$$\mathcal{C}(I) = \text{PREPONE}(L) = closure(L) = ecc(L).$$

∎

6. CONCLUSIONS

We study the relationship of global behavior of composite e-service and local behaviors of the individual e-services in the position. We show that global behavior may sometimes be rather unexpected due to (1) queuing of messages, and (2) distributed decisions made by local peers. Our results indicate that the effect of

combining individual e-services is not very well understood and deserves further investigations. The results also support a top-down approach in developing composite e-services to control/avoid unexpected behaviors.

Acknowledgment: 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.

7. REFERENCES

[1] 3GPP. The 3rd generation partnership project. www.3gpp.org.

[2] 3GPP2. The 3rd generation partnership project 2. www.3gpp2.org.

[3] S. Abiteboul, V. Aguilera, S. Ailleret, B. Amann, F. Arambarri, S. Cluet, G. Cobena, G. Corona, G. Ferran, A. Galland, M. Hascoet, C-C. Kanne, B. Koechlin, D. LeNiniven, A. Marian, L. Mignet, G. Moerkotte, B. Nguyen, M. Preda, M-C. Rousset, M. Sebag, J-P. Sirot, P. Veltri, D. Vodislav, F. Watezand, and T. Westmann. A dynamic warehouse for XML data of the Web. *IEEE Data Engineering Bulletin*, 2001.

[4] S. Abiteboul, V. Vianu, B. Fordham, and Y. Yesha. Relational transducers for electronic commerce. In *Proc. ACM Symp. on Principles of Database Systems*, 1998.

[5] L. D. Alfaro and T. A. Henzinger. Interface automata. In *Proceedings of the Ninth Annual Symposium on Foundations of Software Engineering (FSE'01)*, pages 109–120, 2001.

[6] L. D. Alfaro and T. A. Henzinger. Interface theories for component-based design. In *Proceedings of the First International Workshop on Embedded Software (EMSOFT '01)*, Lecture Notes in Computer Science 2211. Springer-Verlag, 2001.

[7] Philippe Althern. The scala home page. `h` ttp://lamp.epfl.ch/scala/.

[8] A. Ankolekar, M. Burstein, J. Hobbs, O. Lassila, D. Martin, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, and K. Sycara. DAML-S: Web service description for the semantic web. In *Proc. Intl. Semantic Web Conf. (ISWC)*, July 2002.

[9] B. Benatallah, B. Medjahed, A. Bouguettaya, A. Elmagarmid, and J. Beard. Self-coordinated and self-traced composite services with dynamic provider selection. Technical report, University of New South Wales, March 2001. (Available at http://sky.fit.qut.edu.au/dumas/selfserv.ps.gz).

[10] R.V. Book and S.A. Greibach. Quasi-realtime languages. *Mathematical Systems Theory*, 4(2):97–111, 1970.

[11] Business process execution language for web services (version 1.0). http://www.ibm.com/developerworks/library/ws-bpel, 2002.

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

[13] R. Breite, P. Walden, and H. Vanharanta. C-commerce virtuality - will it work in the Internet? In *Proc. of International Conf on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet (SSGRR 2000)*, 2000. (http://www.ssgrr.it/en/ssgrr2000/proceedings.htm).

[14] C. Bussler, R. Hull, S. McIlraith, M.E. Orlowska, B. Pernici, and J. Yang, editors. *Proceedings of Workshop on Web Services, E-Business, and the Semantic Web (WES)*. Springer-Verlag Lecture Notes in Computer Science, number 2512, Toronto, 2002.

[15] F. Casati, S. Sayal, and M. Shan. Developing e-services for composing e-services. In *Proceedings of CAISE 2001*, Interlaken, Switzerland, June 2001.

[16] F. Casati and M.-C. Shan. Dynamic and adaptive composition of e-services. *Information Systems*, 26(3):143–163, 2001.

[17] V. Christophides, R. Hull, G. Karvounarakis, A. Kumar, G. Tong, and M. Xiong. Beyond discrete e-services: Composing session-oriented services in telecommunications. In *Proc. of Workshop on Technologies for E-Services (TES)*, Rome, Italy, September 2001.

[18] G. Cobena, S. Abiteboul, and A. Marian. Detecting changes in xml documents. In *Proc. Int. Conf. on Data Engineering*, 2002.

[19] D. Florescu, A. Grünhagen, and D. Kossmann. XL: An XML programming language for web service specification and composition. In *Intl. World Wide Web Conf. (WWW2002)*, 2002.

[20] S. Gay and M. Hole. Types for correct communication in client-server systems. Technical Report CSD-TR-00-07, Department of Computer Science, Royal Holloway, University of London, December 18 2000.

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

[22] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, 1979.

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

[24] S. A. McIlraith, T. C. Son, and H. Zeng. Semantic web services. In *IEEE Intelligent Systems*, March/April 2001.

[25] S. Narayanan and S. McIlraith. Simulation, verification and automated composition of web services. In *Intl. World Wide Web Conf. (WWW2002)*, 2002.

[26] Simple object access protocol (soap) 1.1. W3C Note 08, May 2000. (http://www.w3.org/TR/SOAP/).

[27] The SIP Forum. Session initiation protocol. www.sipforum.org.