Generalized Third-Party Call Control in SIP Networks

Eric Cheung and Pamela Zave

AT&T Laboratories—Research, Florham Park, New Jersey USA
cheung@research.att.com, pamela@research.att.com

Abstract. Third-party call control (3PCC) is essential to implementing advanced services in Voice-over-IP (VoIP) networks. It allows intermediary applications to control how the media streams of endpoint devices are connected together. However the Session Initiation Protocol (SIP), the widely adopted open standard for VoIP signaling, presents a number of challenges that make 3PCC in SIP complex and prone to errors. Previously proposed solutions only address operations under ideal conditions and thus are incomplete. Furthermore, the problem of compositional 3PCC by multiple applications have not been addressed. In this paper, we propose a general solution for robust and comprehensive media connectivity control. The solution has been verified, and allows multiple applications operating concurrently in a call path to interoperate successfully.

1 Introduction

The widespread adoption of Voice-over-IP (VoIP) creates an opportunity for innovations in new voice, multimedia, and converged (with web and data) services. The Session Initiation Protocol (SIP) is currently the most commonly used open standard for establishing voice and multimedia media sessions. In addition to being an end-to-end signaling protocol, SIP is also useful for invoking intermediary application servers that provide advanced features to the endpoints. For example, at the architectural level the IP Multimedia Subsystem (IMS) is a widely-adopted standard defined by the 3rd Generation Partnership Project (3GPP)[1] for wireless and wireline communications. In the IMS architecture, multiple application servers are invoked one by one based on the profiles of the subscribers, thereby forming a chain of applications between the endpoints. At the level of the application developers, the SIP Servlet Java API (application programming interface) [2] is an emerging standard for programming SIP and converged applications. Within a SIP Servlet container, an *application router* component selects multiple applications to service a call, forming a chain of applications within the container. In both examples, this application chain model may offer several advantages over implementing all applications in the endpoints:

- New applications can be developed as independent modules with standard SIP interfaces, and added incrementally.

- Endpoints only need to support baseline SIP, and do not increase in complexity as more applications are deployed.
- No software installation or upgrade is required at the endpoints. Endpoints may not support such software installation, or it may be operationally undesirable.
- For some applications, executing from the endpoints may impose complex trust and security requirements.
- Endpoints may be turned off or become disconnected from the network. It is easier to maintain centralized servers in the network to be accessible continuously.

One of the key requirements of this model is the ability to control media sessions independently of the media endpoints that actually transmit and receive the media content. Third-party call control (3PCC) refers to this capability. The key characteristics of 3PCC include:

1. An entity that is distinct from the media endpoints can initiate, manage (which may include changing connectivity amongst multiple endpoints, changing media types selection, changing codec selection, and so on), and terminate the media sessions. In this paper, this entity is referred to as the third-party call controller or simply the controller.
2. The third-party call controller has a signaling relationship with each endpoint involved in the media sessions using a standard signaling protocol. The media endpoints do not need to establish and maintain direct signaling relationships amongst themselves, but only indirectly via the controller. Furthermore, from the perspective of each media endpoint the controller appears as a regular peer endpoint, just as in a point-to-point relationship.
3. The media streams flow directly between the media endpoints. The third-party call controller does not access or relay the media streams.

A canonical example of 3PCC is the click-to-dial application. Unlike a regular point-to-point call where the caller initiates the session from his/her endpoint device, click-to-dial allows the caller to initiate the call by clicking on a link on a web page. For example, a company's products page may contain such a link for a potential customer to talk to a sales representative. The HTTP request is translated into a request to a third-party call controller to establish a SIP call to the caller and the callee, and then to set up a media session between them.

Another key requirement of the application chain model is that the 3PCC mechanism must be composable. i.e. the system behavior must be correct even if multiple controllers are in the signaling path, and they operate concurrently. This is crucial to support the goal of modularity.

Third-party call control in SIP has received a significant amount of attention and is recognized as an important call control model. In particular the click-to-dial use case has been cited extensively (e.g. [10], [6], [9]). However, thus far the literature has only included limited example call flows under ideal conditions. Moreover, the click-to-dial use case is relatively simple as the endpoints involved are restricted by the SIP protocol to only a small number of possible actions during call setup time, and only two endpoints are involved. More complex scenarios, for example cases where a controller and the endpoints simultaneously take action causing race conditions, have not been studied in depth. No general and robust solution that can handle error or race conditions has been proposed. Furthermore, composing multiple third-party call controllers have not been studied.

This paper proposes a general solution for third-party call control involving multiple media endpoints at any call states, taking into account race and transient conditions. The solution also supports compositional media control by any number of controllers. Its contributions are twofold: (1) It specifies the correct SIP protocol behavior of a controller under all conditions in order to fulfill the above goals. (2) It divides the functions of a controller into three logical components: the media port, the Hold program and the Flow program, and provides a high-level API for developers to build 3PCC applications that behave correctly and compositionally.

This paper also presents verifications of the safety and correctness properties of the proposed solution. Recommendations on the SIP standard and implementations to facilitate 3PCC are also discussed.

2 The Challenges Presented by SIP

2.1 Summary of SIP Operation

For this paper, we only consider the baseline SIP standard as defined by [13] and any necessary supporting standards, most notably the offer-answer exchange [11]. All other extensions to SIP are excluded and left for future work.

As implied by its name, SIP is primarily designed as a signaling protocol for establishing media sessions. A point-to-point call begins when the caller sends an INVITE request to the callee. The callee can send multiple provisional responses until if it decides to accept the call, it sends a 200 OK response. The caller then sends an ACK message to acknowledge the receipt of the 200 OK response. This INVITE-200-ACK sequence thereby establishes (1) a *SIP dialog* at the signaling level, and (2) a *media session* at the media level, as an offer-answer exchange is conveyed by these messages. The offer-answer exchange reflects the media capabilities of the endpoints, as well as the list of media types the users wish to communicate with. The SIP dialog and the media session persist over the lifetime of the call. Should either party desire to modify the properties of the media session, for example to add video or to temporarily put the other party on hold, it may send a new INVITE request to initiate a new offer-answer exchange. Such mid-call INVITE requests are called re-INVITE requests to distinguish them from the first INVITE that establishes the dialog. Finally, either party may send a BYE request to terminate both the dialog and the media session.

The INVITE-200-ACK messages may convey the offer-answer exchange in one of two ways: (1) INVITE conveys offer, and 200 OK conveys answer (ACK does not contain any session information); and (2) INVITE does not contain offer or answer, 200 OK conveys offer, and ACK conveys answer. These two options are illustrated in Figure 1. In the first option, the initial caller is also the media offerer. In the second option, the initial caller, by not including a media offer in the INVITE, is establishing a SIP dialog but soliciting the callee to make a media offer.

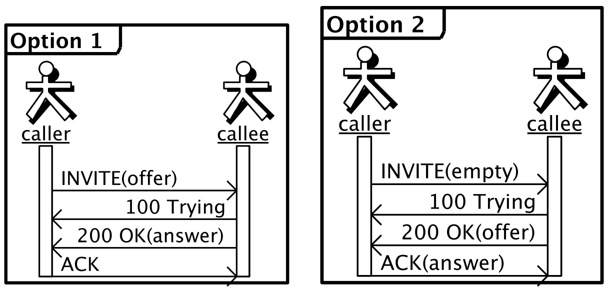

Fig. 1. Two options for media offer-answer exchange in point-to-point SIP calls

A media offer lists the media streams the offerer wishes to include in this media session. The description of each media stream comprises the type of media (e.g. audio or video), the directionality of the stream, and media format information. The media format includes the codecs and other parameters associated with the codec such as video resolution or framerate that the offerer is capable of sending and/or receiving (depending on the directionality). The answerer must send an answer that contains its own media format in response to each media stream in the offer. It cannot include any additional media streams in the answer. The directionality of each media stream must also be compatible with the directionality in the offer. Subsequently, either side may send a new offer that contains additional media streams or change the directionality. The format of the offer and answer conforms to the Session Description Protocol (SDP)[7].

Note: For simplicity, media answers conveyed in provisional responses are not discussed in this paper. As the subsequent 200 OK response must contain the same answer, this omission does not impact generality. Also, for brevity the rest of the paper will omit provisional responses, including the 100 Trying response.

2.2 Issues that Affect 3PCC

A third-party call controller must use the same two options of conveying offer-answer exchange in the INVITE-200-ACK sequence to establish and control the media session between the two actual media endpoints. A click-to-dial call flow is shown in Figure 2, based on 'Flow I' in [10]. Note that there are two distinct SIP dialogs—a dialog between the controller and party1, and another dialog between the controller and party2. The controller uses option 2 to set up the dialog with

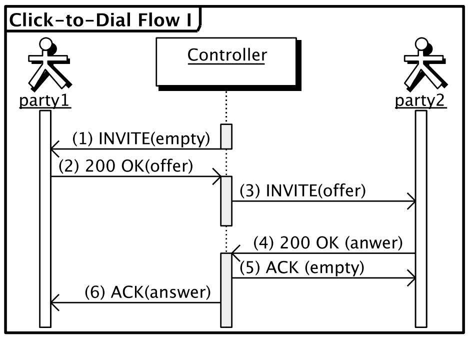

Fig. 2. Example click-to-dial call

party1, and uses option 1 to set up the dialog with party2. By relaying the offer-answer exchange end-to-end, the controller establishes one media session directly between the two parties.

There are several issues with third-party call control in SIP. They are discussed below.

Offer-answer exchange closely linked to signaling protocol

In the signaling layer, the INVITE-200-ACK messages forms a three-way handshake designed for unreliable transports such as UDP. If the callee does not receive the ACK message after sending 200 OK response, it must assume that the response did not reach the caller and retransmit the 200 OK until it receives the ACK. If it does not receive the ACK after a certain timeout (default to 32 seconds), it must consider the dialog and session terminated.

On the other hand, in the media layer if the 200 OK contains an offer, the ACK must contain an answer. However, the answer may not always be immediately available. Consider the click-to-dial call flow in Figure 2. After the controller receives message (2) 200 OK(offer) from party1, it must relay the offer to party2, and wait for the answer before it can send (6) ACK together with the answer to party1. If party2 is not an automated system but a real person, it may take up to a few minutes for it to send (4) 200 OK(answer). Therefore, this delay may result in unnecessary retransmission of messages, or in the worst case lead to failure to establish the media session.

Offer can only be answered once

There are many use cases where a controller first connects the caller to one party, then after a period of time connects the caller to a second party. For example, in a calling-card application the controller may first connect the caller to a prompt-and-collect media server to authenticate the caller and to obtain the destination. When that is completed the controller calls the destination. It is natural that the media offer from the caller is applicable equally to the offer-answer exchange with the media server, and the offer-answer exchange with the final destination. A naive controller may store the offer from the caller, and send it to the final destination after the prompt-and-collect phase is completed. However, in SIP offer-answer, it is not possible to send more than one answer to an offer. Another naive solution is to send the second answer to the caller as an offer. However, if the answer from the caller is different from the initial offer, the the controller must then again send this answer as an offer to the final destination. This may result in an infinite offer-answer spiral. dwet As a result, once the controller has sent an answer to an offer, it cannot reuse the offer and send it to another endpoint because there is no way to send an answer to the initial offerer.

Reduced media and codec choices

When an endpoint responds to a media offer, the answer must correspond to the offer and cannot add new media streams. While it may add codec choices to streams already present in the offer, or subsequently send a more full-featured offer to add media streams or include additional codecs for existing streams, the answerer usually would not do so. After all, the offerer has already indicated the media types and codecs it desires to communicate with. As a result, the answerer usually does not have an opportunity to make an unilateral declaration of its full capabilities and desires.

Another common situation is that after the initial offer-answer exchange is completed, the two endpoints may perform another offer-answer exchange to narrow down the media or codec choices. For example, many endpoints support a multitude of codecs, but can only handle one codec at a time. Such an endpoint would send an offer listing the supported codecs. Once the peer endpoint has indicated the codecs it can support in the answer, the first endpoint would then send another offer with just one codec to lock it down.

These two properties of the offer-answer exchange affect 3PCC. When a controller connects two endpoints together, even if it has an outstanding unanswered offer from one endpoint, it must not send it to the other endpoint as a fresh offer. Doing so may restrict the codec and media choices that the two endpoints may communicate with.

One outstanding INVITE transaction at a time

The SIP protocol dictates that after a UA has sent a re-INVITE request, it must wait for the transaction to complete before it can send another re-INVITE request. Similarly, the offer-answer protocol dictates that an offerer after sending an offer must wait for the answer before sending another offer. This means that if a controller performs media connectivity switching in a state where a re-INVITE or an offer has been sent, it must wait for the answer to return before proceeding. This asynchrony adds complexity and additional states to the application logic of the controller.

Re-INVITE request glare

When two SIP user agents (UAs) are in an established dialog, both of them may send mid-dialog re-INVITE requests. If they send re-INVITE requests at the same time, a glare condition arises. The SIP protocol dictates that each UA must then reject the re-INVITE request with a 491 Request Pending response. Each UA may try to send the re-INVITE request again, but if the UA is the original initiator of the dialog (i.e. the initial caller), then it must do so after waiting 2.1-4 seconds. If the UA is initially the callee, then it must do so after waiting 0-2 seconds([13] Section 14.1). Thus if both UAs decide to retry, the initial callee would be able to do so first, and the second attempt should not result in glare.

The possibility of re-INVITE glare creates additional complexity for a third-party call controller. When in the middle of a media switching operation the controller sends a re-INVITE request, it may be rejected. Even when the controller is not in the middle of media switching but is simply acting as an relay, the two endpoints may send re-INVITE at the same time, causing glare at the controller.

In our experience, the re-INVITE glare causes significant difficulties and complexity to a robust 3PCC design.

SDP manipulation

The third-party call controller connects two previously unconnected endpoints together by causing a media offer-answer exchange to occur between the two endpoints. However, the SIP protocol mandates that the session descriptions in an offer-answer exchange within a SIP dialog must conform to certain restrictions. Firstly, the answer must contain the same media streams as the offer. Secondly, any new offer must retain the number and order of media streams from the previous exchanges, although new streams may be added. Thirdly, the origin line in the SDP sent to an endpoint must remain unchanged, except for the version number which must be incremented by one in each successive SDP sent.

While this manipulation is not technically hard, it requires the controller to constantly parse, modify and format SDP in SIP messages as they are relayed between endpoints.

2.3 Existing Solutions in SIP Community

The SIP community has worked on 3PCC since the early days of the protocol. The often cited Best Current Practice document [10] is the culmination of at least four years of work by the IETF SIP and SIPPING working groups. Its main focus is on how a third-party call controller can initiate a call to two parties as in the click-to-dial use case. It studies four candidate call flows, and recommends two of the flows for different circumstances. It also suggests a call flow that switches one party in a two-party call to a mid-call announcement while putting the second party on hold, and then reconnects the two parties after the announcement is completed.

However, this document falls short of offering a complete solution. In all the call flow examples, the endpoints are quiescent and the controller is the only SIP entity that initiates signaling transactions, and thus all the call flows may proceed in an orderly fashion. This is helped by the fact that for the click-to-dial use case endpoints are not allowed to send any requests until a SIP dialog is established. However, in general and especially for mid-call 3PCC, the controller may receive SIP messages at inconvenient times and must be able to handle them. For example, the endpoints can send re-INVITE messages on their own. Thus when the controller is triggered to perform mid-call 3PCC operations, it may be in the middle of re-INVITE transactions with one or more of the endpoints. The document addresses one such case but the proposed handling of the scenario is potentially problematic (see “Generate dummy offer or answer” section in 3.1). Furthermore, the document only discusses cases where there is only one controller in the call path. It does not address how the media controlling actions from multiple controllers can be composed in a correct manner such that the desired end result may be achieved. In general, the controllers operate independently of each other, and thus they are likely to perform media action at the same time, causing various race and glare conditions.

Nonetheless, [10] provides many useful ideas and strategies. The authors have made use of and expanded many of these ideas in formulating our generalized solution.

3 Proposed Solution

3.1 General Strategy

This section can be viewed as a collection of basic tools that a third-party call controller may make use of. Then in the following section, a more structured design of the controller is proposed, making use of many of the same tools.

Solicit an offer with empty INVITE request

This technique is based on ‘Flow I’ in [10], and solves the second and third problems discussed in Sections 2.2. The controller can solicit a media offer from an endpoint by sending an INVITE request that contains no SDP. This is a basic pattern in most 3PCC call flows. In Figure 2, messages 1, 2 and 6 represent this pattern. Note that there is a dependency on the endpoint to send its full set of media capabilities and the user’s desired set of media streams in the offer conveyed in the 200 response, even if it is already in the middle of a SIP dialog and there is a media session established. Its offer cannot be restricted by the offer-answer exchange that has already taken place. This dependency is discussed in Section 7.1.

Create a no-media SIP dialog first

In the first issue discussed in Section 2.2, the dual-purpose of the ACK message to acknowledge receipt of 200 OK and to convey an media answer causes a dilemma of having to send the ACK quickly, and having to wait for the media answer. This problem is solved by a new call flow called Flow IV in [10], and which is illustrated in Figure 3. The controller first establishes a SIP dialog and a session with zero media streams with the first party (messages (1) to (3)). Once the dialog is established, the controller can use the same call flow in Figure 2 (but exchange party1 and party2). Because party1 has already answered the initial INVITE request, it is assumed that it will respond to (6) re-INVITE and provide an answer in (7) 200 response immediately. Therefore the controller can send (9) ACK to party2 quickly to avoid excessive retransmission or timeout. Note there is dependency on the endpoint to respond quickly to mid-dialog re-INVITE requests. Section 7.2 discusses this dependency in full.

Flow I shown in Figure 2 is still useful in cases where the second party is known to respond quickly, for example if it is an automaton like a media server. In such cases, it is not necessary to create the no-media SIP dialog first.

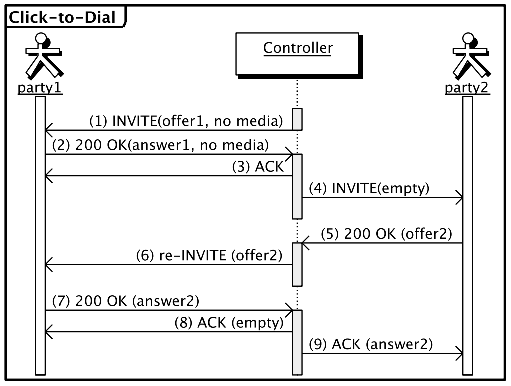

Fig. 3. Click-to-Dial SIP call, revised (Flow IV)

Generate dummy offer or answer

As discussed in Section 2.3, if the endpoints are not quiescent, the controller must handle situations where the endpoint sends a re-INVITE request in the middle of the controller's operation. One instance of this occurrence is discussed in [10]. Consider the call flow in Figure 4: after controller sends (4) INVITE, party2 may take some time to respond. In the meantime, party1 sends a re-INVITE with a new offer. How should the controller respond so it can proceed with its operation when party2 responds with the answer? The recommendation in [10] is to send a 491 Request Pending response to reject the re-INVITE. However, this approach

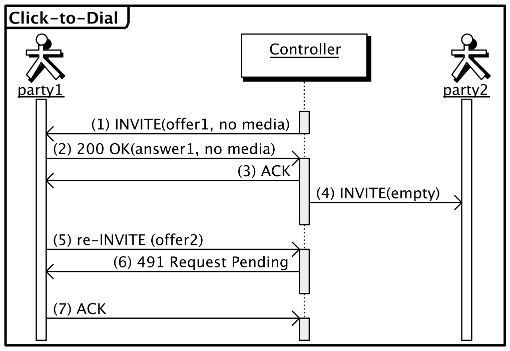

Fig. 4. Click-to-Dial Flow IV rejecting unexpected re-INVITE

may cause a situation where both sides repeatedly resend re-INVITE requests, all of which could then be rejected by the other side, causing an infinite retry loop. Thus the controller will fail to complete the call flow and to connect the two parties together. This is particular problematic if party1 is in fact another controller.

We propose that a better alternative is to send an answer which disables each offered media stream by setting the port in the answer to zero. This is appropriate as the controller is not ready to connect party1 to any media endpoint yet. Similarly, if party1 sends a re-INVITE with no SDP to solicit an offer, the controller can send a dummy offer which is a SDP with no media stream.

Handle re-INVITE glare

A similar situation arises when the controller and an endpoint send re-INVITE at the same time, creating the glare condition discussed in Section 2.2. The controller must follow the retry strategy according to the SIP specification. If the controller is the initiator of the SIP dialog, it may receive a retry of the re-INVITE before it is allowed to retry its own. In this case, the controller may have to respond with 200 OK with a dummy offer or answer as discussed in the previous section.

It is important that the controller does not create a re-INVITE glare knowingly. For example, if it has received a re-INVITE from an endpoint and has not responded to it yet, the controller must not send a re-INVITE to that endpoint. Instead, it must respond to the re-INVITE first before starting a new transaction.

Maintain state of offer-answer exchange

As discussed in Section 2.2, if the controller is in the middle of a re-INVITE transaction and the associated offer-answer exchange, it must wait for it to complete before it can initiate another one. For example, if it has sent an offer in a 200 OK response, it must wait for the answer in the ACK. This necessitates tracking the state of the offer-answer exchange in the SIP dialog with each endpoint. For the purpose of storing the media state of a SIP dialog, we do not use the actual SIP message names. Rather, we use abstractions of the messages that indicate their media-related content. This affords the advantage that the abstraction can still apply when in the future we consider other SIP extensions that may also convey media offer-answer exchanges, for example the reliable provisional response extension[12].

Within the baseline SIP protocol, the two ways in which the INVITE-200-ACK messages can convey media offer-answer exchange can be abstracted as follows:

Option 1		Option 2	
INVITE(sdp)	offer	INVITE	solicit
200 OK(sdp)	answer	200 OK(sdp)	sol_offer
ACK	–	ACK(sdp)	answer
491 Request Pending			error

Table 1. Abstract media protocol messages

3.2 Design of the Controller

At the high level, a third-party call controller manages the media connectivity amongst a set of endpoints. There is a SIP dialog between each endpoint and the controller. We shall refer to the controller end of such a dialog as a *media port* or simply a port. Depending on application logic, the controller may isolate an endpoint (put on hold), or link two endpoints together in a media session. This suggests that its operation may be divided into two functional programs: the *Hold* and the *Link* programs. As the controller changes the way endpoints are connected, it puts their corresponding media ports into the appropriate Hold and Link programs. The Hold and Link programs examine the states of the media ports, and send SIP requests and responses accordingly to achieve the goal of the programs.

As an example, Figure 5 shows a call-waiting application (CW) where subscriber S is in two separate calls with A and B. At the top of the figure, the subscriber is connected to A, and B is put on hold. CW puts the ports for S and A in a Link program, and the port for B in a Hold program. When the subscriber signals to CW to switch the lines (e.g. by a flash signal), CW puts the port for A in a new Hold program, and puts the ports for S and B in a new Link program.

A media port is responsible for tracking the state of the offer-answer exchange in its corresponding SIP dialog. A state transition occurs whenever a SIP message that contains media semantics is received or sent on the port.

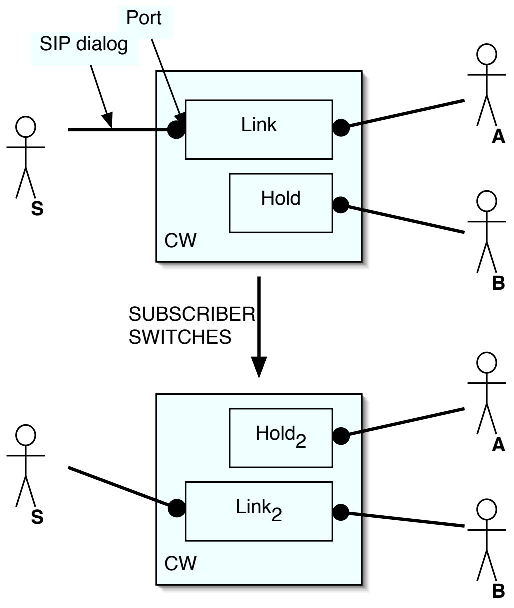

Fig. 5. 3PCC in a call-waiting application

Each media port is always in a Hold or Link program. When a port receives a SIP message that contains media semantics, it passes the message to the program it is in. When the program wishes to send a SIP message to the endpoint, it passes the message to the port to send out. The Hold and Link programs may decide to send more than one message at a time. In the case of the Link program, it may also decide to send one or more SIP messages on the other port.

The following sections discuss the details of the media port and the Hold and Link programs.

Media port finite state machine

As the controller exchanges SIP messages with an endpoint, the state of the corresponding media port changes. This is represented by the finite state machine (FSM) shown in Figure 6. The abstract media protocol messages in Table 1 are used. The state names that start with C_ refer to states in which the port is acting as UA client in the current transaction, i.e. it has sent the INVITE or re-INVITE. Conversely, the state names that start with S_ refer to states in which the port is acting as UA server in the current transaction. A state transition that is labeled !message means the port is sending the message, and a state transition that is labeled ?message means the port has received the message. When either the endpoint or the controller terminates the SIP dialog by sending a BYE request, the FSM is destroyed.

While most of the FSM should be self-explanatory, a few details are worth discussing. When the media port is in the C_SOLICIT_SENT or C_OFFER_SENT

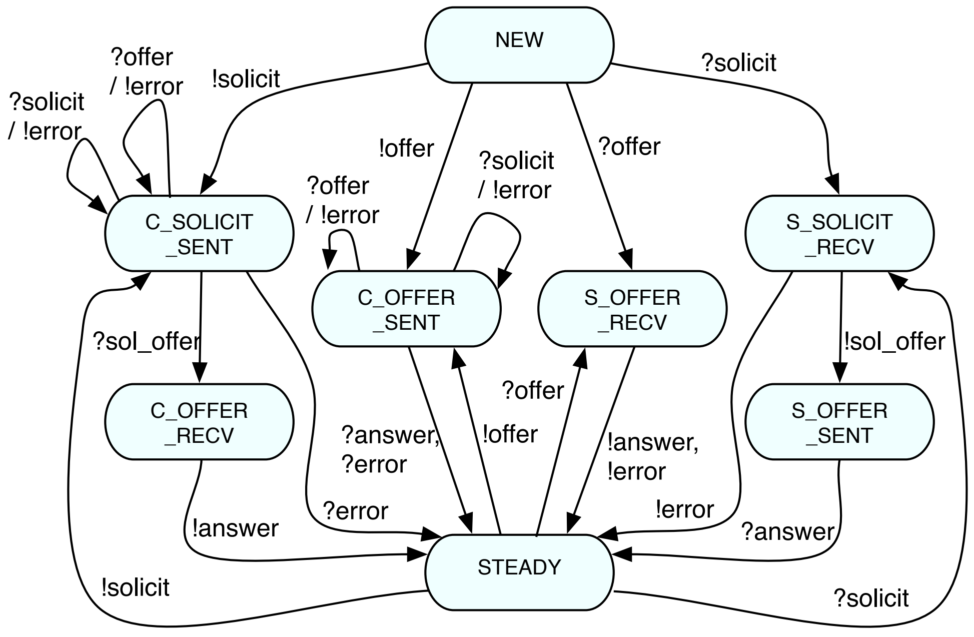

Fig. 6. Finite state machine of a media port

state, it may receive a solicit or offer from the endpoint due to re-INVITE glare. The media port always sends an error response (denoted for example by ?solicit/!error in the state transition). Typically the media port will then receive an error response from the endpoint, and transition to the STEADY state.

From the S_OFFER_RECV or S_SOLICIT_RECV state, the Link program may decide to send an error on this port because it has received an error from the other port. The port transitions to the STEADY state. This case will be discussed in the Link program section below, and illustrated in Figure 7(b).

The Hold program

The Hold program operates on a single media port. Its function is relatively simple—to stop the endpoint from transmitting media. If there is an established SIP dialog, the Hold program must send a re-INVITE to disable all existing media streams by setting the port number of each media stream to 0. If the endpoint sends a new offer, the Hold program must generate an answer that disables all media streams. Finally, if the endpoint sends a solicit, the Hold program must send an offer that disables all media streams.

If upon entry the port state is such that the Hold program cannot send a re-INVITE request (e.g. C_OFFER_SENT), it must wait until a response is received at the port and the state transitions to STEADY before sending the re-INVITE request. On the other hand, if upon entry the media port is in certain states, the Hold program can take short-cut. For example, from S_OFFER_RECV the Hold program can send an answer that disable all media streams immediately.

After sending a re-INVITE request, the request may be rejected if the endpoint is also sending a re-INVITE causing a glare condition. If the incoming re-INVITE contains an offer, the Hold program can simply sends an answer that disables all media streams, instead of sending its own media offer.

The Link program

The Link program operates on a pair of media ports, thus its function is much more complex. The Link program operates in two phases: (1) the matching phase, and (2) the transparent phase.

Upon entry, the Link program begins operation in the matching phase, the goal of which is to match the media states of the two ports. First, it examines the states of the ports. If both ports are in the NEW state, the Link program immediately enters the transparent phase. Otherwise, depending on the port state the controller may have to send the appropriate dummy offer or answer to complete any outstanding transactions. For example, if a port is in the C_OFFER_RECV state, the Link program can send a dummy answer and transition to the STEADY state. It may also need to wait for a message to arrive at a port. For example from the C_OFFER_SENT state the Link program must wait for an answer to arrive and transition to the STEADY state.

Once the states of the two ports are suitable, the Link program can send a solicit message on one port to start the basic end-to-end offer-answer exchange call flow. As above, this solicit message may be rejected due to glare. If the Link program loses the re-INVITE retry then it must handle the incoming re-INVITE and wait till the media states are suitable again.

Similar to the Hold link, if upon entry one of the ports is in certain media states the Link program can take shortcuts. For example, if the state of port1 is C_SOLICIT_SENT upon entry, then the Link program does not need to send a solicit because one is already sent. The Link program will need to make sure that port2 is in STEADY or S_SOLICIT_RECV state, then upon receipt of an sol_offer on port1 it can send an offer or sol_offer respectively on port2.

Once the Link program has solicited and relayed end-to-end a fresh offer-answer exchange (because the offer is solicited, it contains full media capabilities) between the endpoints, it enters the transparent phase. In the transparent phase, the Link program passes media offers and answers transparently between the endpoints in most cases. However, it is important that the Link program does not create glare by sending a re-INVITE to an endpoint after it has received a re-INVITE from that endpoint. This is illustrated in Figure 7. In (a), the Link program passes re-INVITE from party1 to party2 (message 1 and 2). At the same time, party2 sends re-INVITE (3). If Link program simply passes it to party1 (4), it would create a glare in the dialog with party1. Furthermore, this creates new media states in both dialogs, which adds significant complexity to both the Hold and the Link programs as they will now need to handle these new states on entry. In contrast, the handling shown in (b) is simpler. The Link

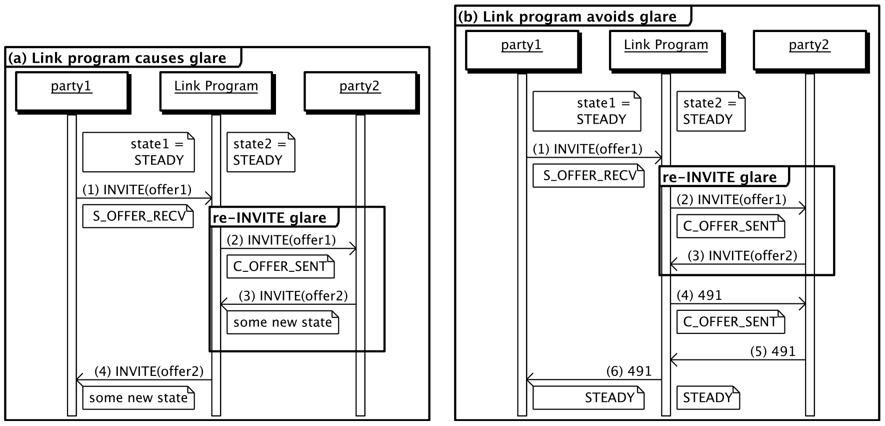

Fig. 7. Re-INVITE glare in Link program

program rejects the re-INVITE (3) with a 491 Request Pending response (4), and keep the media state in C_OFFER_SENT. If subsequently party2 sends a 491 response (5), the Link program can pass it to party1 (6), returning both dialogs to the STEADY state.

If a re-INVITE is rejected for reasons other than glare, the Link program must notify the application logic. For example, an endpoint may reject an offer because it cannot accept any of the media streams. How such error conditions should be handled depends on the specification of the application. For example, the application may decide to abandon the switching operation and return to the media connectivity prior to the operation. In other cases, the application may decide to tear down the call altogether.

Because with 3PCC the two endpoints may both be initial callers or initial callees, they may retry the re-INVITE within the same time interval (i.e. 2.1–4 or 0–2 seconds respectively). The SIP specification suggests that the UA wait a random time inside the specified interval, but we have encountered UA implementations that always retry after a fixed interval. This issue is discussed in Section 7.3.

3.3 Detailed Operation

The call-waiting scenario shown in Figure 5 will be used to demonstrate some of the operations of the Link and Hold programs. A message sequence diagram is shown in Figure 8. Initially ports S and A are in Link1. As Link1 is in transparent state, it relays (1) offer from S to A (2). At this point the subscriber signals CW to switch to B. CW destroys Link1, and creates Link2 and Hold2. Ports S and B are put in Link2 in S_OFFER_RECV and STEADY states respectively. The Link program sends (3) answer to S to respond to the pending offer and put S on hold, and sends (4) solicit on port B to get a fresh offer. An end-to-end offer-answer exchange then takes place from (5) to (8), and the Link program reaches

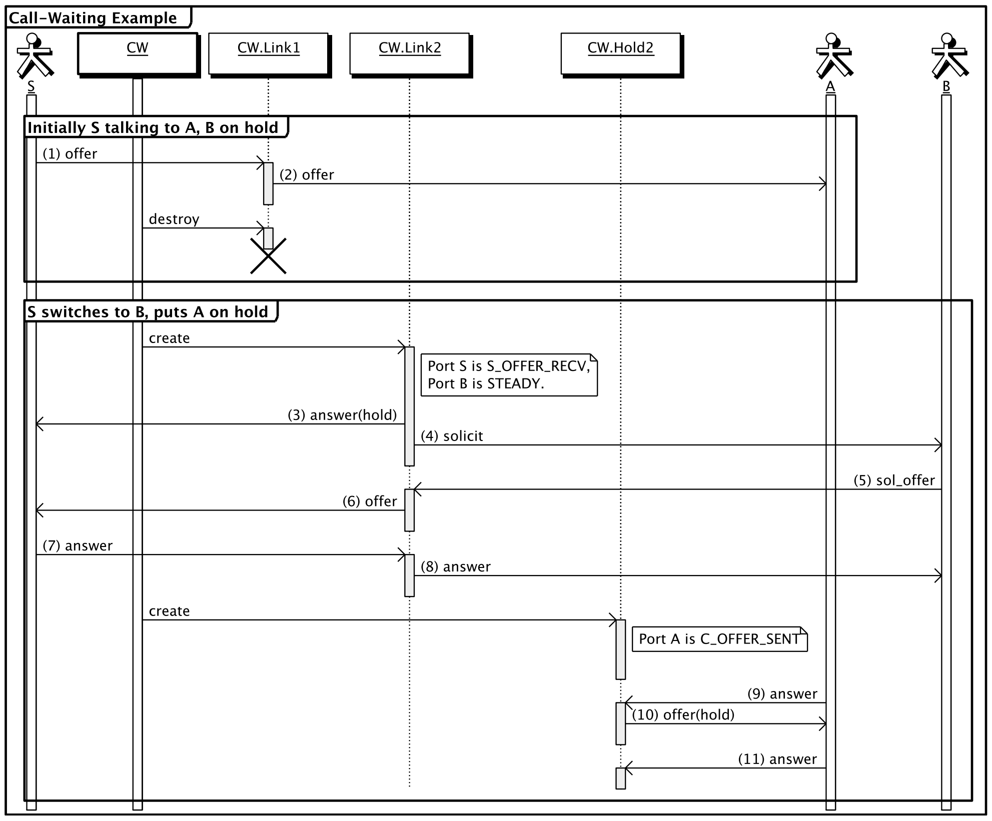

Fig. 8. Message sequence diagram of a call-waiting scenario

transparent phase. At the same time, port A is put in Hold2 at C_OFFER_SENT state. The Hold program must wait for (9) answer before sending (10) offer to put A on hold.

4 Formal Verification

In order to verify the correct behavior of the proposed solution, we have developed a model of the controller in the specification language Promela. This model includes the Hold program, the Link program and the media port. The abstracted media messages, i.e. solicit, offer, sol_offer, answer, and error, are used in the model. A Promela model of a SIP UA was also developed to serve as the environment in which the controller operates. The UA model is cooperative, and always responds positively to solicit or offer except under glare conditions. We then use the formal verification tool Spin[8] to verify various configurations of the models. For brevity, we only discuss the verification of the Link program as it is more complicated than the Hold program.

The Link program and the UA are modeled as concurrent processes that execute independently. Messages are passed between processes via unidirectional FIFO *channels*. At any given state of the model, each process may have multiple possible execution steps. Spin executes one step from amongst all possible steps in all the processes, and advances the model to a new state. This step-wise execution is repeated, resulting in an execution *trace*. If the trace reaches a state where there are no possible execution steps, the execution stops at an end state. In an exhaustive search, Spin explores all traces, i.e. all possible interleaving of execution steps. All verifications performed for this paper are exhaustive searches. They are divided into two parts as discussed below.

4.1 Protocol Conformance and Correctness

In order to verify that the Link program conforms to the SIP and offer-answer protocols, and that it correctly establishes a media session between the two endpoints, we use the safety test features of Spin coupled with appropriate instrumentation in the model. Spin raises an error if one of the following conditions occur during a search:

Invalid end states An execution trace must not terminate when a UA is in the middle of an offer-answer exchange. The UA model defines a valid end state as one where there is no pending offer or answer.

Assertions Numerous assertion statements are included in the models. When a media port sends or receives a message it asserts that it is allowed in the current state. When a UA sends a media offer, it includes a new version number. When it subsequently receives an answer, it asserts that the version number matches that in the offer. When the Link program enters the transparent phase, it asserts that it has relayed an end-to-end offer-answer exchange between the two UAs.

Non-empty channels An execution trace must not stop when there are still unhandled messages in the FIFO channels. This is an indication that a message arrives when the model does not expect it.

In addition, Spin also performs coverage analysis and lists statements unreached in any of the traces.

The first configuration connects two UA models directly. This is shown in Figure 9(a). Each UA model can send any one of all allowable messages at a given state. For example, initially a UA has the option to send either solicit or offer, or do nothing. This purpose of this analysis is to validate the UA model. The coverage analysis also verifies that all statements in the model are reached, indicating that all possible UA actions are modeled.

In the next configuration in Figure 9(b), two UA models are connected to the Link model. At the beginning of execution, both media ports are in NEW state, thus the Link program enters the transparent phase immediately. This configuration validates that the Link program in the transparent phase conforms to protocols, and that it correctly relays the offer-answer exchange between the two UAs, allowing them to establish a media session.

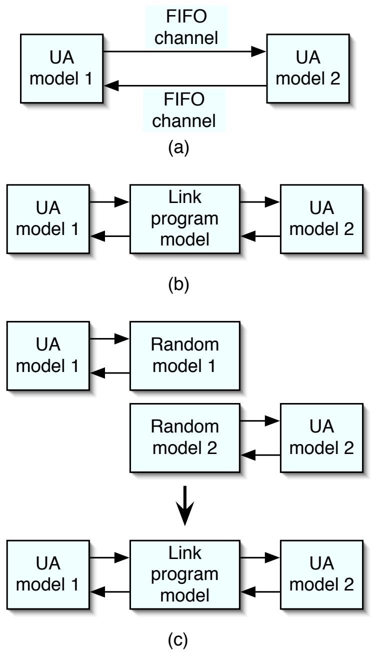

Fig. 9. Configurations of Promela models for Spin verification

To validate the entire Link program, we first develop a Random program. The Random program operates on one port, and is connected to a UA model. From each port state, the Random program either sends out one of the allowable messages or does nothing. The configuration in Figure 9(c) first connects each UA to a Random program and allows some execution to occur. This causes the media port to reach any one of its states. Then the UAs are connected to the Link program. Thus the Link program starts operation when the two media ports and the messaging channels are in arbitrary but legal states. The coverage analysis verifies that across the traces the Link program is entered with all combinations of port states.

In general, the Link program begins in the matching phase. Through the use of assertion statements, it is verified that when the Link program progresses to the transparent phase it has solicited an offer from one of the UA models, and has completed an end-to-end exchange of that offer and corresponding answer between the two UA models.

4.2 Liveness Property

We have not yet verified that the Link program always succeeds in reaching the transparent phase. In order to verify this property, a linear-time temporal logic (LTL) verification is used. The LTL formula that expresses this property

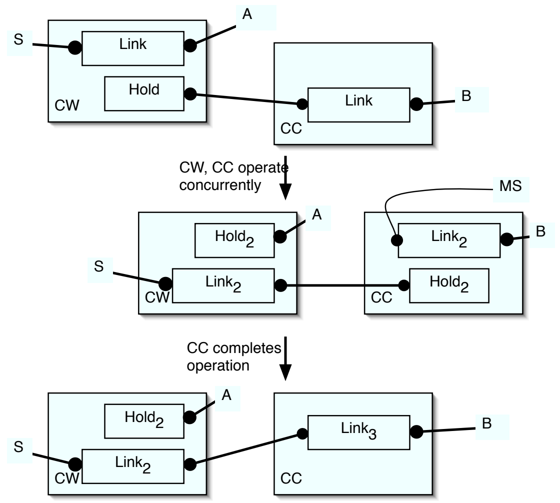

Fig. 10. Composition of two applications

formally is:

$$\Box\ (isLinked \rightarrow \Diamond\ isTransparent)$$

where *isLinked* is set to true when the UA models are connected to the Link model, and *isTransparent* is true when the Link program enters the transparent phase. This formula says that in any verification trace, it is invariantly true ($\Box$) that connecting the two UAs to the Link program implies ($\rightarrow$) that eventually ($\Diamond$) the Link program will reach transparent phase.

The UA model must be modified for the liveness verification. If a UA, upon receiving solicit or offer from the Link program, always sends its own solicit or offer to cause a glare, the Link program will be flooded and can never reach transparent state in such a hostile environment. Therefore, the UA model is modified to send only a finite number of offer or solicit messages. However, after the UA has received an offer or solicit message from the Link program and has sent a positive response, the count is reset so it can send those messages again.

A successful Spin exhaustive search verifies that in a non-flooding environment the Link program always reaches the transparent phase.

5 Compositional Media Control by Multiple Controllers

An important goal of this work is to support multiple controllers in the call path between the media endpoints. This is a very common scenario, and often the controllers belong to different administrative domains and as such there cannot be any overarching framework that coordinates their operations. For example, consider again the call-waiting example but this time B subscribes to a calling-card (CC) application. As shown in Figure 10, while CW is triggered to connect

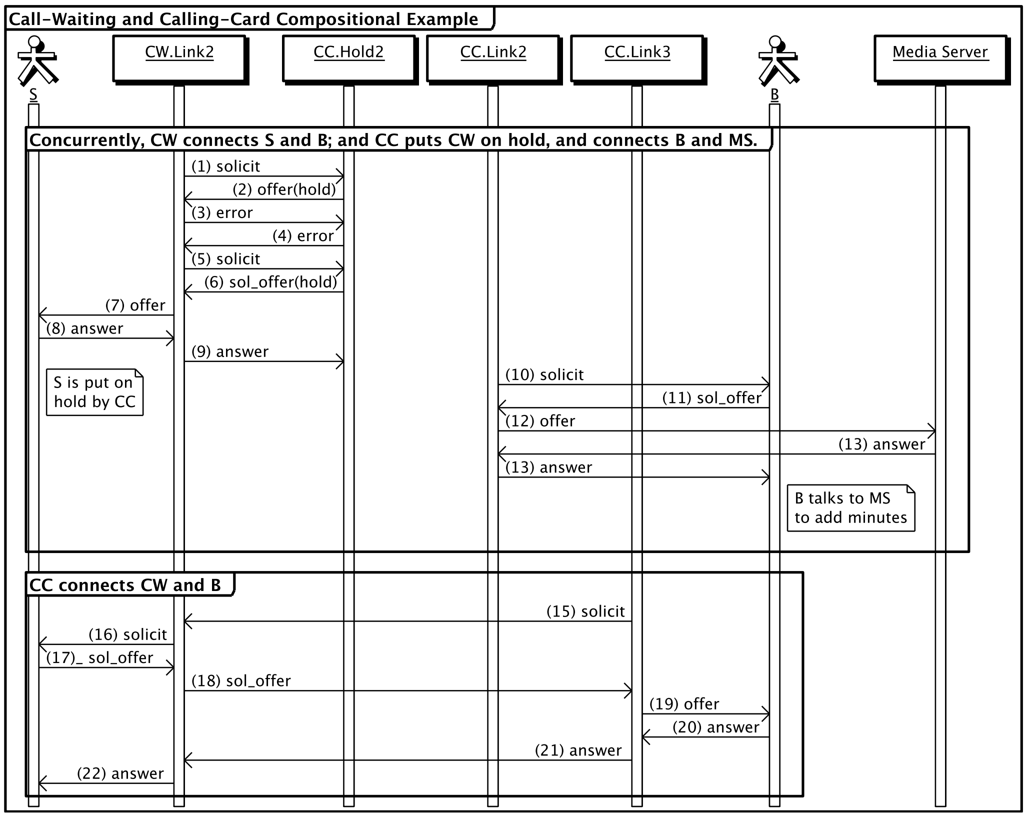

Fig. 11. Message sequence diagram of a call-waiting and calling-card scenario

S and B, CC may also operate at the same time to connect B to a media server to request payment.

Such simultaneous operations must not lead to deadlocks, infinite loops, or incorrect end states. Rather, the controllers must cooperate to arrive at the correct media connectivity of the endpoints. The Hold and Link programs are designed to support compositional 3PCC by multiple controllers. Figure 11 shows a message sequence that the programs might generate while implementing this scenario. The glare caused by the concurrent CW and CC operation and its resolution are shown by messages 1 to 9. In this case because CW was the initial callee in the CW–CC dialog, CW can retry sending the solicit message first. At the same time, CC connects B with a media server. When the payment collection is completed, CC reconnects B to the call towards S. Because CW has entered transparent phase past message 9, it simply relays the offer-answer exchange between CC and S.

It is impossible to use Spin to verify configurations of all possible number of controllers in a chain because the problem is unbounded. Instead, we are working on a proof by induction to shown that the proposed solution converges with any number of controllers.

6 Application of the Solution

The generalized solution presented in the paper offers a concrete blueprint for SIP developers when programming 3PCC applications. With the correct placement of media ports in Hold and Link programs, applications can correctly perform media control operations under any transient or race situations.

The structure of this approach lends itself to familiar software development goals of modularity, abstraction and reuse. Clearly the media port FSM and the Hold and Link programs can be implemented as reusable software libraries or objects. Application developers would then have a high-level, *goal-oriented* API for safe and robust media connectivity control. The media-affecting SIP messages and the state of each port can be abstracted and hidden from the programmer (although the programmer can still have visibility to them if necessary because sometimes application logic depends on them). In most use cases, the application logic is triggered to alter the media connectivity by an asynchronous event, for example a user action or a timer expiry. When this occurs, the programmer can express the new media connectivity goal by putting the appropriate ports into newly created Hold and Link program objects. The programmer may do so without being concerned with the current port states. The program objects will then perform their tasks to fulfill the goals expressed by the programmer. This may take some time but their operation can be hidden from the programmer.

The authors and their colleagues are currently working on implementing this solution using the open source ECharts for SIP Servlets development kit [14]. ECharts for SIP Servlets is a state-machine-driven programming environment for the SIP Servlet API. In this environment, the media state FSM and the Hold and Link programs can be implemented as *machine fragments* that can be re-used by other higher level ECharts for SIP Servlets machines.

7 Recommendations for the SIP Community

This generalized 3PCC solution has a few dependencies on SIP UA behavior that are not currently mandated by the SIP specifications. These dependencies have been pointed out in Section 3, and will be discussed fully in this section. The authors propose that the SIP community consider these issues and their impact on correct 3PCC operations, and if deemed appropriate establish best current practices for UA implementations or modify the SIP specifications.

7.1 Solicited Offer Contains Full Media Capabilities

In order to achieve full media capability exchange, 3PCC has a dependency on UA behavior. When a UA receives a mid-dialog media solicit (i.e. re-INVITE request with no SDP), it should send an offer (i.e. 200 OK response to the re-INVITE) that reflects its full set of media capabilities and the user's desire. This is a SHOULD strength specification in [13] Section 14.2: "A UAS providing an offer in a 2xx (because the INVITE did not contain an offer) SHOULD construct the offer as if the UAS were making a brand new call, ... Specifically, this means that it SHOULD include as many media formats and media types that the UA is willing to support."

Implementers of SIP UAs should be aware of the impact if an implementation deviates from this behavior. It may lead to inability to establish media connection with the new UA to which the controller connects it due to lack of common codecs. It may also lead to reduced media type even though both UAs support it, for example if a UA does not offer video because the UA it was previously connected to only supports audio.

7.2 Delayed Response to Re-INVITE

This solution has a dependency on a UA to respond to mid-dialog re-INVITE requests quickly. In Figure 3, party1 must respond to (6) re-INVITE(offer2) quickly. Otherwise, the controller cannot send (9) ACK to stop party2 from retransmitting (5) 200 OK(offer2). If party2 does not receive an ACK after it completes all its retransmissions (by default after 32 seconds), [13] section 14.2 suggests that it should terminate the dialog: "If a UAS generates a 2xx response and never receives an ACK, it SHOULD generate a BYE to terminate the dialog." Therefore, the results may range from increased network traffic due to retransmissions, to users being disconnected.

We have tested several UA implementations and they do not alert the user of the new offer in the re-INVITE, and do respond quickly. However, it seems reasonable that some implementations may elect to consult the user, especially if the offer adds a new media type. Indeed, [13] suggests in section 14.2: "If the session description has changed, the UAS MUST adjust the session parameters accordingly, possibly after asking the user for confirmation."

If the user does not respond quickly, then this problem will be exposed. We propose that this should be a subject for discussion within the SIP community on how this may be resolved.

7.3 Re-INVITE Glare Retry Timer

This dependency affects how two UAs connected through a controller handle the re-INVITE glare condition. The glare resolution mechanism in [13] is based on a backoff-retry strategy. However, the randomness of the retry timer is not mandated. Section 14.1 says: "If a UAC receives a 491 response to a re-INVITE, it SHOULD start a timer with a value T chosen as follows: ... If the UAC is not the owner of the Call-ID of the dialog ID, T has a randomly chosen value of between 0 and 2 seconds in units of 10 ms."

We have encountered UA implementations that use a fixed timer value. Such implementations are still compliant because this is a SHOULD strength clause, and also the clause does not specify that a new random value must be generated every time. If a click-to-dial application connects two such implementations, and subsequently they simultaneously send re-INVITE, then the glare will not be resolved as they will retry at the exact same time.

We propose that the SIP specification strengthen this clause, or establish best current practices for UAs to calculate a new random timer value every time.

8 Related Work

The work in the SIP community on 3PCC is mainly documented in [10]. Its contributions and limitations have already been discussed throughout this paper. The authors are not aware of any previous published work on compositional 3PCC by multiple SIP-based controllers, perhaps because the problem is not yet widely recognized. Therefore we have to look beyond SIP for relevant related work.

In the public switched telephone network (PSTN), a telephone call typically traverses multiple telephone switches. Each of them may perform media switching. However, in the PSTN the signaling and media travel along the same path and media switching is performed locally within each switch. Therefore compositional control is straightforward.

Distributed Feature Composition (DFC) is an architecture for composing feature logic modules (*feature boxes*) in a pipe-and-filter manner. An IP-based implementation of DFC [4] addresses compositional media control by using a central media-control module [5]. The central media-control module maintains a graph that represents media connectivity amongst the external endpoints and the feature boxes. When a feature box performs media switching, it sends commands to the central media-control module, which in turn updates the graph. Based on the graph, the media-control module can then determine how endpoints are connected to each other. This implementation has been deployed as the advanced feature server for a nationwide consumer VoIP service [3]. However, this approach has two obvious deficiencies: (1) the centralized algorithm is a performance bottleneck, and (2) if the feature modules reside in multiple administrative domains, they must have a signaling and trust relationship with the central media-control module.

In order to improve on the central graph-based algorithm approach, the authors have successfully developed a comprehensive solution for distributed control [16]. This solution includes an architecture-independent descriptive model, a new media-control protocol, a set of high-level programming primitives, a formal specification of their compositional semantics, and an implementation. This solution has been partially verified for correctness, and analysis shows that it offers better performance than a comparable SIP-based solution. However, because the solution relies on a new, non-SIP protocol designed specifically for compositionality, it is not directly usable in existing SIP networks.

The work presented in this paper draws heavily on the experience and insights gained while working on the previous solutions. Comparing the solution reported in [16] to the solution for SIP here, the SIP solution is more complex. It is also less efficient in the sense that it requires more message exchanges to accomplish the same media switching operation, especially under glare conditions. However, it can be applied directly to SIP deployments in the many service provider networks and enterprises today.

Moving to a higher level of abstraction, the Parlay Group [15] specifies the Parlay/OSA and the Parlay X APIs for IT developers to access multimedia communication networks such as SIP. The Parlay/OSA API supports 'multiparty call control', which covers click-to-dial and more generally managing multiple parties and changing the media connectivity amongst them. The Parlay X API supports 'third party call' which is a subset of the Parlay API capabilities. These are high-level, protocol-agnostic APIs. A SIP network may expose these APIs to applications via a Parlay gateway. As such, the Parlay standards do not specify how these media control operations in the APIs may be supported at the SIP level, but leave the details to each Parlay gateway implementation. The solution presented here is well-suited for implementing correct 3PCC behavior in Parlay gateways.

9 Conclusion and Future Work

In this paper, we have presented a comprehensive and generalized solution for third-party call control in SIP networks. The solution enables endpoints to establish maximum media capabilities and types, and it can also handle transient states and race conditions that can occur in the SIP protocol. The solution supports compositional control whereby multiple third-party call controllers are in the call path and perform media control operation simultaneously. Finally, the solution has been verified for correctness and liveness properties with a model checking verifier when a single controller is involved.

This solution has several dependencies on SIP UA behavior that are not strictly mandated by the SIP specification, although they are commonly adhered to by existing UA implementations. These dependencies have been identified and recommendations are made for their resolution.

Some ongoing and future works include the proof by induction discussed in Section 5, and adding support for selected SIP extensions, for example the reliable provisional response RFC [12]. Work is also underway to implement the solution using the ECharts for SIP Servlets development environment. When completed this will greatly facilitate developing SIP Servlet-based third-party call controllers.

Acknowledgments. Michael Jackson made important contributions to our earlier work on media control. This work has also benefited from our long term collaborations with Greg Bond, Hal Purdy and Tom Smith.

References

1. 3rd Generation Partnership Project. TS 22.228: Service requirements for the Internet protocol (IP) multimedia core network subsystem. Technical report, 3GPP, December 2007. V8.3.0.
2. BEA. SIP servlet API version 1.1, 2008. Java Community Process JSR 289. http://jcp.org/en/jsr/detail?id=289.
3. G. W. Bond, E. Cheung, H. H. Goguen, K. J. Hanson, D. Henderson, G. M. Karam, K. H. Purdy, T. M. Smith, P. Zave, and J. C. Ramming. Experience with component-based development of a telecommunication service. In *Proceedings of the Eighth International SIGSOFT Symposium on Component-Based Software Engineering*, pages 289–305. Springer-Verlag LNCS 3489, 2005.
4. G. W. Bond, E. Cheung, H. Purdy, P. Zave, and J. C. Ramming. An open architecture for next-generation telecommunication services. In *ACM Transactions on Internet Technology*, volume IV, pages 83–123, February 2004.
5. E. Cheung, M. Jackson, and P. Zave. Distributed media control for multimedia communications services. In *IEEE International Conference on Communications*, volume 4, pages 2454–2458, 2002.
6. T.-C. Chiang, V. K. Gurbani, and J. B. Reid. The need for third-party call control. *Bell Labs Technical Journal*, 7(1):41–46, 2002.
7. M. Handley and V. Jacobson. SDP: Session description protocol, April 1998. IETF RFC 2327.
8. G. J. Holzmann. *The Spin Model Checker: Primer and Reference Manual*. Addison-Wesley Publishing Company, 2004.
9. R. Mahy, R. Sparks, J. Rosenberg, D. Petrie, and A. Johnston. A call control and multi-party usage framework for the session initiation protocol (SIP). IETF Internet-Draft draft-ietf-sipping-cc-framework-09, November 2007.
10. J. Rosenberg, J. Peterson, H. Schulzrinne, and G. Camarillo. Best current practices for third party call control (3pcc) in the session initiation protocol (SIP), April 2004. IETF RFC 3725.
11. J. Rosenberg and H. Schulzrinne. An offer/answer model with the session description protocol (SDP), June 2002. IETF RFC 3264.
12. J. Rosenberg and H. Schulzrinne. Reliability of provisional responses in the session initiation protocol (SIP). RFC 3262, June 2002.
13. J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session initiation protocol, June 2002. IETF RFC 3261.
14. T. M. Smith and G. W. Bond. ECharts for SIP Servlets: a state-machine programming environment for VoIP applications. In *IPTComm '07: Proceedings of the 1st International Conference on Principles, Systems and Applications of IP telecommunications*, pages 89–98. ACM, 2007.
15. The Parlay Group. Homepage. http://www.parlay.org/.
16. P. Zave and E. Cheung. Compositional control of IP media. To appear in IEEE Transactions on Software Engineering, 2008.