Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Alexander Knauf

Scalable, Distributed Conference Control in Tightly Coupled SIP Scenarios

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Alexander Knauf

Scalable, Distributed Conference Control in Tightly Coupled SIP Scenarios

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Technische Informatik
am Studiendepartment Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Thomas Schmidt
Zweitgutachter: Prof. Dr. Franz Korf

Abgegeben am August 25, 2009

Alexander Knauf

Thema der Bachelorarbeit
Scalable, Distributed Conference Control in Tightly Coupled SIP Scenarios
Stichworte
SIP, Gruppenkonferenzen, Verteilte Konferenzkontrolle, Peer-to-Peer Systeme, Konferenzstatus Ereignisse
Zusammenfassung
Mit dem Session Initiation Protocol ist es möglich spontane Gruppenkonferenzen zu erzeugen. Um solche Konferenzen zu koordinieren wird eine kontrollierende Instanz benötigt, Focus genannt, die bestimmt nach welchen Parametern die Medienströme übertragen werden sollen und welche Geräte für das Verteilen der Medienströme zuständig sind. Solche Focus Instanzen können bei steigender Anzahl an Konferenzteilnehmern überlastet werden und schlechte Antwortzeiten erbringen. Ein Lösungsansatz für das Problem der Lastkonzentration besteht darin, die kontrollierende Instanz auf mehrere unabhängige Systeme zu verteilen, dabei jedoch keine externen Protokolle zu verwenden und das Verteilungsproblem für die Konferenzteilnehmer transparent zu gestalten. Diese Arbeit verfolgt den Ansatz durch Aufteilung der Rollen zwischen Identifier und Locator einer SIP Konferenz URI, sowie einer Erweiterung des Conference Event Packages zur Sicherstellung der Konsistenz des Status der verschiedenen Focus Instanzen, ein Basisprotokollschema für verteilte Konferenzkontrolle zu liefern.

Alexander Knauf

Title of the paper
Scalable, Distributed Conference Control in Tightly Coupled SIP Scenarios
Keywords
SIP, Tightly Coupled Conference, Distributed Conferencing, Peer-to-Peer Systems, Conference Event States
Abstract
Tightly coupled group conferences created with the help of the Session Initiation Protocol need to be controlled by an entity called focus which accepts and handles SIP calls and controls the media mixers. Such Focus entities are challenged by an increasing number of conference participants, causing a possible decline in signaling performance. A solution for this problem is to distribute the controlling instance on multiple independent systems. It is desirable to achieve this without using external protocols and to keep the distributed conference control transparent to clients. This work elaborates on the concept of splitting the role of identifier and locator of the conference URI and by extending the SIP conference event state package to ensure a uniformly consistent view at all conference members.

Contents

1 Introduction

1.1 Motivation

An increasing number of broadband Internet connections with flat rates offer an alternative to traditional telephony and create a market for network conferences. These techniques known as Voice over IP (VoIP) or Video and Voice over IP (VVoIP) allow general costumers calling friends and family, arranging business meetings via the Internet or offering a so called "Teamspeak" service for multiuser online gaming.

Some of these conferencing applications are based on closed, proprietary protocols and are not applicable to large size conferences. The voice and video quality of Instant Messenger (IM) applications is often rather limited. These applications are frequently only able to establish simple point-to-point communications. Skype [1], a popular multi-point IM client, allows users to establish voice and video calls to other users running Skype, or enable dial-out to traditional telephones in good quality. Skype is able to create two-point video conferences for up to 25 participants in a voice multiparty [43]. Its disadvantages are due to the closed Skype protocol [4] and limitation of conference members size. Other applications are using open communication protocols like the Session Initiation Protocol (SIP) [39] or H.323 [18]. Various organizations support video conferencing using these open protocols by providing a Multipoint Control Unit (MCU) to distribute the media streams to all joining conference members. In those scenarios, an organization offers a reliable service for media conferencing to its clients, but has to be pay for the infrastructure. Because of the high complexity of processing multiple streams at the media level and distributing them in transport layer, these MCU systems are of limited scalability. In particular an increasing number of participants and maintaining multiple simultaneous conferences may case overloading of an MCU causing worse voice and video quality at the endpoints. For a lightweight ad hoc conferencing solution it is desirable to remain independent of expensive hardware or restrictive infrastructure, but nevertheless, providing a viable conferencing service.

To approach this challenge, various peer-to-peer deployments show how an overlay network can handle millions of joining members at low cost of memory per node and keep short routing distances. Concepts like Skype or Peer-to-Peer SIP (P2PSIP) [5] already use an overlay network for data storage and resource location, independent of central servers. Only some login servers or bootstrap nodes are needed to establish such an overlay. Both systems have resolved the location discovery using their overlays, but comply to the traditional conferencing concept of using a single node, which is responsible for controlling the media session and distributing the media streams simultaneous.

The distribution of conference control and media mixing is already a working area in the Internet Engineering Task Force (IETF). Those approaches concentrate on creating additional protocols for communication among conference controllers sharing conference spe-

cific informations, but maintaining transparency of the distributed conference content to their clients.

The motivation for this work is to present a solution for a hardware structure independent conference scenario, by creating a base protocol schema for distributed conference control. This will be achieved by using an open and extensible protocol in a Peer-to-Peer environment and thereby providing a viable scalable service.

1.2 Overview of this Work

Establishing spontaneous network conferences is now a common method for communication throughout the Internet. It extends the options to stay in contact with other persons and bridge the distance to them. A traditional intercontinental phone call can be very expensive, why general costumers are searching for alternative ways. The success of IM like ICQ, Yahoo Messenger or Skype demonstrates the demand of this growing market. Conference systems deployed within professional organizations save costs of frequent business trips and allow for meetings on demand.

This work targets at multimedia conferences that may reach large size and operate without an MCU. It illustrates the concept of distributing conference control among a set of equal nodes within the conference members provide a module for Scalable Distributed Conferencing (SDCON) [24] in a transparent fashion to SDCON-unaware participants. This approach is based on SIP as an open signaling protocol and does not need additional protocols for synchronizing conference information among the controlling entities.

It adapts automatically to the number of participants joining the conference, and achives constant delay times to participate a multiparty session. SDCON provides mechanisms to delegate incoming SIP calls to previous discovered conference controllers in reason to balance the management effort among them. A synchronization mechanism for these controlling entities is achieved by an extension of the conference event package, making every SDCON node aware of the current state of the conference.

Splitting the conference control in a SIP-based approach produces the conflict that multiple controlling entities have to appear as one single, logical entity to the participating clients. This work solves this conflict in a SIP-conform manner, so that standard compliant parties are not exposed to change requirements.

1.3 Organization of the Report

This work is organized as follows.

Chapter 2 begins with an historical overview of the development of video conferencing systems in the past followed by a short presentation of modern network conferencing. It will be presented a general introduction to conferencing with the Session Initiation Protocol. Starting with a basic introduction to the Session Initiation Protocol, common approaches to

conferencing in so called tightly coupled scenarios will be explained by summarizing related work in this area.

After the introduction, this work presents the problem statement and concept of scalable distributed conference control in section 3. The challenges for such a distributed scenario will be explained and the solution elaborated in this work will be presented.

Chapter 4 engages with the subject of the so called 'Event Packages' and their design in XML format. It presents a document schema for conference state information and will extend this schema by adding multifocus issues.

Having summarized the theoretical base of the functionalities for conferencing with SIP, it will lay out the implementation of the proposed scheme in chapter 5. It shortly introduces the programming languages and libraries used, followed by the design of the framework behind this work explain the detailed implementation by providing some source code snippets.

The next chapter 6 is dedicated to measurements and evaluations. Setups and results that verify the concept of scalable distributed conferencing is operational are outlined along with performance results obtained from emulations that have been performed in practice.

A conclusion and outlook chapter 7 will conclude this report, summarize results and show directions for future work in this area.

2 Group Conferencing

This chapter introduces general group conferencing, and in particular discusses conferencing in SIP Scenarios.

2.1 History and Introduction

The possibility to communicate per voice and video via the Internet is not matter of course. Every analogue voice and video signal recorded by cameras or microphones will be converted to digital signals, encapsulated in datagrams and routed throughout the Internet to a destination where it may be decoded and rendered to the end user. However, this whole process poses high requirements onto the end systems and appropriate network quality to perform the video call in soft realtime.

The first analogue video call presented at the Internationale Funkausstellung (IFA) 1929 in Berlin. It consisted of two closed-circuit television systems connected via cable and were used in some German Reich Postamt (Post Office) in Berlin from 1930 - 1934 [30]. In the 1960s the AT&T started expensive research to develop a videophone for replacing traditional telephony. The result was a video telephone named 'Picturephone' [14] which used an analogue signal at 1 MHz Bandwidth and had a $5 * 5\frac{1}{2}$ inch screen size. Due to very low demand of this product, the service faded away in the middle of the 1970s and was a commercial failure. One of the first digital video telephony solutions was made possible by the development of Integrated Services Digital Network (ISDN) with compressed video and audio transmission in the 1980s. Another approach of the year 1981 was the Packet Video Protocol (PVP) as an extension for the Network Voice Protocol (NVP-II) [10] that consisted in a data protocol for transmission of video data. An promising protocol was developed in the Internet Stream Protocol (ST/ST2) [44, 12] at the IP layer. It intended to provide more end-to-end real-time guarantees over the Internet that would have make media streaming more effective. ST/ST2 were already reserved the for name 'IPv5' at the IANA, but the deployment of this protocol would have produced such high costs,that it has been declined for economic reasons. In the 1990s IP based video conferencing became possible by new compression technologies and using personal computers (PC) as communication medium.

Since the turn of the millennium hardware capacities, a wide availability of high bandwidth connectivities and the development of highly efficient video compression techniques [17] facilitate video conferences that allow multiparty conversations accessible for everyone. The continuous development of mobile devices with powerful processors and integrated cameras allows video conferencing in transit and presents the state of the day of ubiquitous digital voice and video interaction [11, 29].

Today conferencing scenarios exist in three types of models, Fully Distributed, Loosely Coupled and Tightly Coupled conferences. In fully distributed multiparty conferences, all participants have a pairwise relationship in signaling and media mixing to every other participant

may in a full mesh topology. There is no central point of control to which the conference members could synchronize and no possibility to retrieve conference information. This architecture implies a limited scaling behavior, because N participants require signaling and mixing in the order of $O(N^2)$ complexity.

Loosely coupled scenarios also not require a central point of control and all signaling is handled by the conference members. It is realized with lightweight sessions that do not explicit indicate a conference membership [28]. This approach differs from the fully distributed model by using multicast media groups for media stream publishing and may offer additional services like automatic source selection or lip synchronization to the conference end systems. Many routers in the Internet do not support multicast groups, so that loosely coupled conferences are hardly practicable on a global scale.

The tightly coupled conference model consists of one or several conference controlling entities to which the end systems can explicitly apply for membership. The conference controller is the central point of control for all signaling messages to its participants and may perform media mixing as well. It should negotiate the capabilities by gathering the properties of the end systems including their supported media types and codecs. Commonly it performs a negotiation phase before starting a conference. Open standards that support media negotiations are H.264 [17] or the Session Description Protocol (SDP) [16], exchanging information of each joining end system in an offer/answer model [38]. The conference controller processes these information to reach an agreement and informs every participant about the conferences media properties. Another challenge to a conference controller may results from floor control at a running conference. It may be responsible to decide which end system is allowed to send media data at a given time.

Shown below are some general requirements for group conferences [28]:

- Reliability: detect failures and fix them if possible
- Scalability: sustain quality with increasing members sizes
- Security: provide authentication and encryption
- Efficiency: keep overhead small latency low
- Simplicity: allow for easy implementation

This work approach attempts to design a solution for scalable distributed conference control. It aims to meet most of these requirements by using SIP [39] as a lightweight signaling protocol in tightly coupled scenarios.

2.2 Conferencing with SIP

The Session Initiation Protocol (SIP) is an application layer control and signaling protocol for establishing, modifying or terminating sessions for point-to-point or multi-point commu-

nication. SIP messages can be used to carry session description information which allow participants to agree on a set of compatible media types. It is independent of the lower transport layer protocol types and can be extended to meet additional requirements.

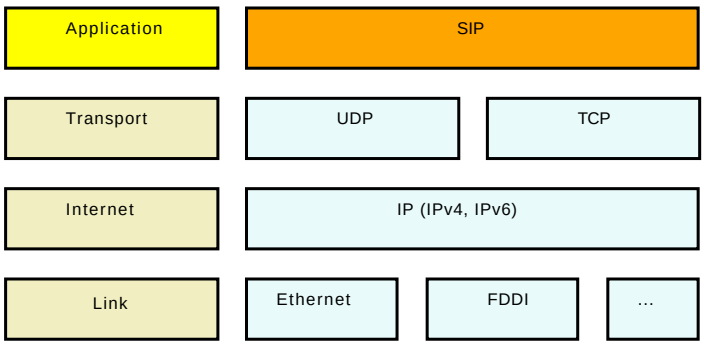

Figure 1: SIP with the DoD Reference Model

SIP supports five functionalities for multimedia communication [39]:

- *User location*:

  To locate an endpoint system, called User Agent (UA), SIP enables an infrastructure of network hosts to which the UA can register. These hosts, called proxies, will route further SIP Requests to their destination by for example DNS lookup.

- *User availability*:

  The Session Initiation Protocol offers multiple types of response messages to indicate the willingness of the called party endpoint to engage in communication.

- *User capabilities*:

  SIP Messages can carry application information about supported media types and media parameters.

- *Session Setup*:

  Caller and callee are enabled to establish a media session after parameters have been exchanged.

- *Session management*:

  Established sessions can be modified, transfered to other endpoints or terminated.

A SIP dialog is a peer-to-peer relationship between two user agents that persists for some time. A set of SIP requests generate a SIP dialog by receiving a positive response from the called UA. The most typically dialog occurs from a SIP INVITE request which initiates a SIP

call. Figure 2 shows an example of a SIP call between two user agents (Piggy and Kermit) routed through SIP their proxies. Piggy is registered at the registrar server `muppets.com`. Her SIP identity is a composition of her user name and the domain both form a SIP Uniform Resource Identifier (URI). In this case, it is `sip:piggy@muppets.com`. Piggy sends a SIP INVITE to Kermit via her outbound proxy. This proxy has the responsibility to discover the location of the callee `sip:kermit@sesamestreet.com` by resolving the domain and to route the message to the inbound proxy at the destination of `sesamestreet.com`. The inbound proxy at `sesamestreet.com` forwards the INVITE to Kermit. Kermit, willing to accepted the call, sends confirmative responses. Responses of type 1xx and 2xx change the dialog state to an 'early' dialog. To confirm the dialog (and change state to 'confirmed'), Piggy sends an ACK Message. The session will be terminated by sending a SIP Bye request.

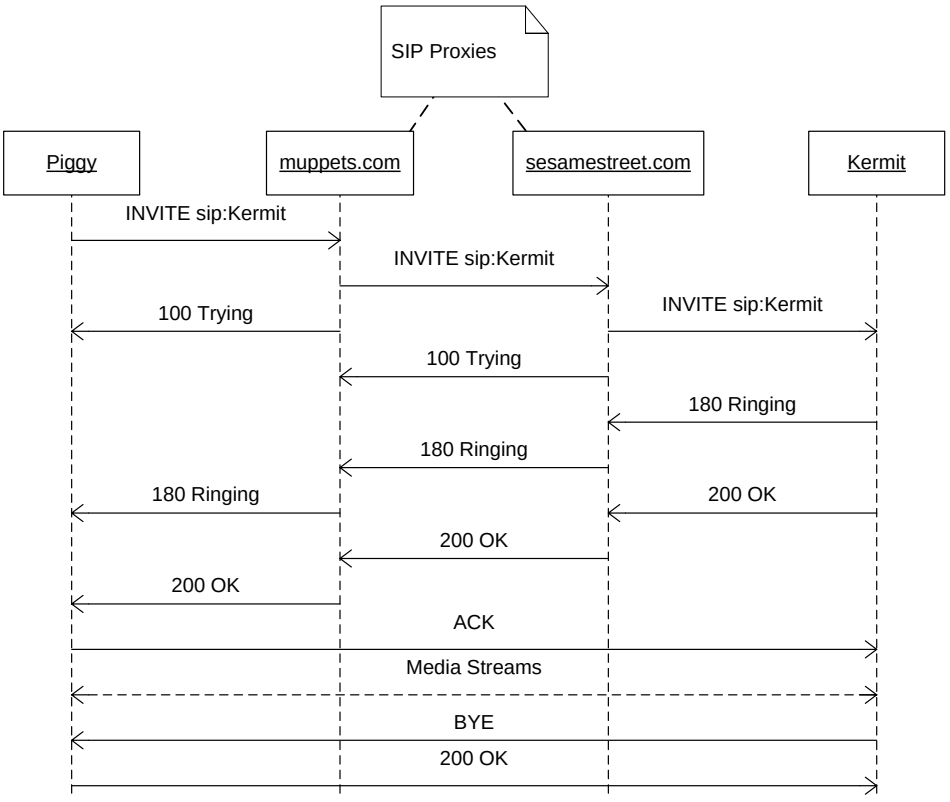

Figure 2: Call setup with SIP trapezoid

The SIP invite request shown in figure 2 could look like this:

```
INVITE sip:kermit@sesamestreet.com SIP/2.0
Call-ID: 0815@141.22.26.6
CSeq: 1 INVITE
From: "Piggy" <sip:piggy@muppets.com>;tag=134652
To: "Kermit" <sip:kermit@sesamestreet.com>
Via: SIP/2.0/UDP 141.22.26.6:5060;branch=z9hG4bKf1
Max-Forwards: 70
Contact: <sip:piggy@141.22.26.6>
Content-Length: 159
```

A tightly coupled SIP conference with multiple participants is an association of SIP user agents with one central point [25]. The central point of a conference is the focus [35]. The focus is a SIP user agent that can be addressed by a unique SIP URI exclusively foreseen for a specific conference. Figure 3 shows the functionalities provided by a SIP conference application and shows the focus as a component of it. It has direct relationships to all participants, maintaining every single SIP dialog to them, and can be implemented either by a conference participant or a separate application server. The focus is responsible for all the media receipting members be to served by using one or more media mixers. Mixers combines incoming streams to one or more outgoing media streams. Each conference may have a conference policy that describes how a conference should operate, e.g. which user agents are allowed to join the conference. The policy could be set up by a participant using non-SIP mechanisms. The focus has access to this policy and needs to enforce it. Another duty of the conference controller is to handle and publish the current conference state to its participants. This includes user agents' join or leave of the conference. Such event state mechanisms are achieved by sending notifications the to participants which subscribed the notification service [31].

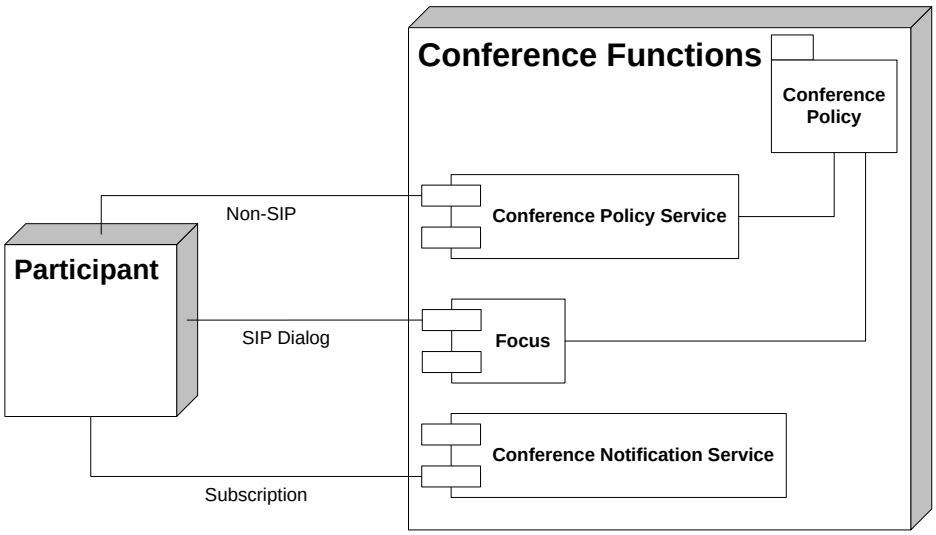

Figure 3: Overview of conference functionalities

2.2.1 Joining a Conference

There are multiple methods to join a SIP conference that are published in extensions RFCs to SIP [39]. The addition of new participants can be controlled by first party (a user agent adds himself) or third party (a user agent adds another user agent). The most common way of joining is done by sending a SIP INVITE request to the conference URI (first party) [35]. The URI could be learned through many mechanisms, for example via e-mail, instant messenger or published on a web page. If the conference policy allows the request, the focus will send a 200 OK response and includes the user into the conference. To end the conference membership, the participant sends a SIP BYE request, as if it were a normal call.

It is also possible to ask the focus for inviting a user agent into an existing conference by sending a SIP REFER request extension [41] (third party). A SIP REFER request includes a Refer-To header field carrying a SIP URI of the user agent to be invited. If the focus accepts, it will send a SIP INVITE to the requested UA indicating its role by an `isfocus` parameter in the Contact header field. A REFER request implies a subscription to the refer event. For example, a focus may has to notify the requester of a pending invitation or whether the invitation was successful. The REFER request can be sent outside of a SIP dialog created by an INVITE request and generates a new dialog between the two endpoints. This request can be also sent directly to the party that is pleased to join the conference. The Refer-To header then refers to the conference URI the participant is asked to join to. This assumes

that the requested peer is able to understand the REFER method. It would produce a failure response otherwise.

Another method to join a conference or a two-party conversation is defined by the SIP extension SIP Join Header [26]. A SIP INVITE request, addressing one of the conference endpoints, includes a join header which contains the call-id for the existing SIP dialog the new participant wants to join. The use cases of a join header to enter a conference instead of sending an INVITE request to the conference URI are as follows:

- The INVITE request is trying to join a two-party conversation. In this case, a new conference must be initiated.
- The calling user agent is unaware of joining a conference instead of a dialog.
- The new participant may not know the conference URI but the call-id

A sample call flow is shown in figure 4. Piggy is in a SIP conversation with Kermit with the call-id `4711@kw`. Elmo decides to join the conversation by sending a SIP INVITE to Kermit. The request includes a join header field with the call-id information for the established dialog. Elmo obtained the information by non-SIP-mechanisms. Kermit accepts the call and issues a re-INVITE to Piggy. Now the two-party call has change to a multiparty conference. A more detailed explanation about re-INVITE and conference initiation is presented in the next section.

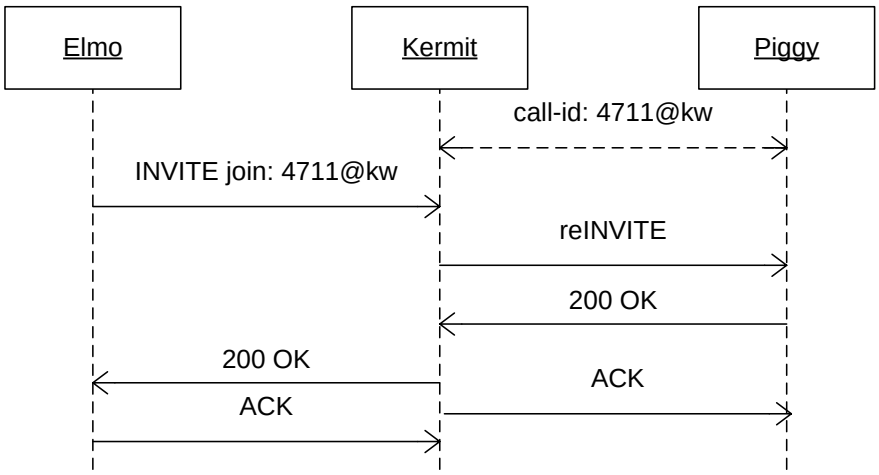

Figure 4: SIP join header example

2.2.2 Conference Initiation

The instantiation of SIP conferences need to distinguish in two typical scenarios:

1. The conference focus is deployed on an application server:

   Some organizations are providing a dedicated server infrastructure to offer SIP conferences (controlling and media mixing) to its clients.

2. Conferences are realized in peer-to-peer scenarios:

   A user agent has implemented a module for focus capabilities to establish ad hoc conferences.

In both scenarios, a focus endpoint has to identify itself as a conference controller by adding the `isfocus` tag in the contact header field, unless it explicitly wants to hide [1] its focus abilities [22]. To a participant that understands the `isfocus` addition is called a conference aware user agent. It can use this to setup call control conventions to the multiparty session or subscribe to notifications services to receive conference specific informations [31]. Participants which do not understand the `isfocus` tag may simply ignore it and just dial into a conference as if it were a normal call. These UA are called conference-unaware.

It is fundamental that a conference focus is identified by a Globaly Routable User agent Uniform resource identifier (GRUU) [36]. A conference URI is always a SIP or SIPS [2, 13] URI and might be opaque to any participants using it. Only from the looked at the URI, a participant cannot decide whether it belongs to a conference or any other service. This goes in concordance with general usage of URIs as described in [7]. Conference URIs can be long-lived to represent interest groups with a focus always available to manage conference issues to participants currently joining or short-lived for ad hoc multiparty conversations.

For conferences controlled by a dedicated server, RFC 4579 [22] presents some call flow examples that are used, lightly modified, for ad hoc multiparty sessions for the framework beyond this work. The original call flow examples will not be discussed in this work.

In the SIP community exists a great interest in peer-to-peer call mechanisms to create ad hoc multiparty conferences. The RFC 5359 [23] gives example Call flows to create a multimedia session.

3-Way conferences:

- Third party is added to the conference:

  Figure 5 displays a call scenario, in which Piggy invites Kermit to a session.First Kermit, who is capable of media mixing and multiparty call control, wishes to add Elmo to the conversation. The corresponding INVITE request from Kermit to Piggy is also called a re-INVITE because it is issued within an existing session. It is used to setup

[1] Hiding focus abilities is no aspired behavior in conferencing, but possible

an existing session and may include new descriptions for multiparty use. The re-INVITE includes a new Contact header carrying a conference-specific URI, in this example `puppets@sesamestreet.com`, and indicating the conference content by the `isfocus` tag. After Piggy confirmed the re-INVITE, Kermit invites Elmo to join the multimedia session by sending an INVITE including the same modified information to the conference.

The re-INVITE to Piggy may look like this:

```
INVITE sip:piggy@muppets.com SIP/2.0
Call-ID: 0815@141.22.26.6
CSeq: 1 INVITE
From: "Kermit" <sip:kermit@sesamestreet.com>;tag=134652
To: "Piggy" <sip:piggy@muppets.com>;tag=654131
Via: SIP/2.0/UDP 141.22.26.6:5060;branch=z9hG4bKf1
Max-Forwards: 70
Contact: <sip:puppets@sesamestreet.com>;isfocus
Content-Length: 167
...
```

- Third party initiated using Join header:

  A point-to-point session may change to a multiparty conversation by a third party call arriving at one of the endpoints. The INVITE received could carry a Join header corresponding to figure 4. As callee provides the capabilities to serve as a focus, it firstly accepts the call by responding with 180 Ringing and re-invites the user agent that it was already in the dialog by sending an INVITE request including a the new conference contact URI plus `isfocus` tag. Finally the called user agent confirms the dialog to the third party.

- Alternative third party initiated invite:

  An alternative variant to handle a third party initiated multiparty conversation arises from avoiding the Join header field. SIP implementations may not support this header e.g. like the NIST JAIN SIP [19] stack used by the framework behind this work. It fully comprises to SIP standards presented in RFC 4579 [22].

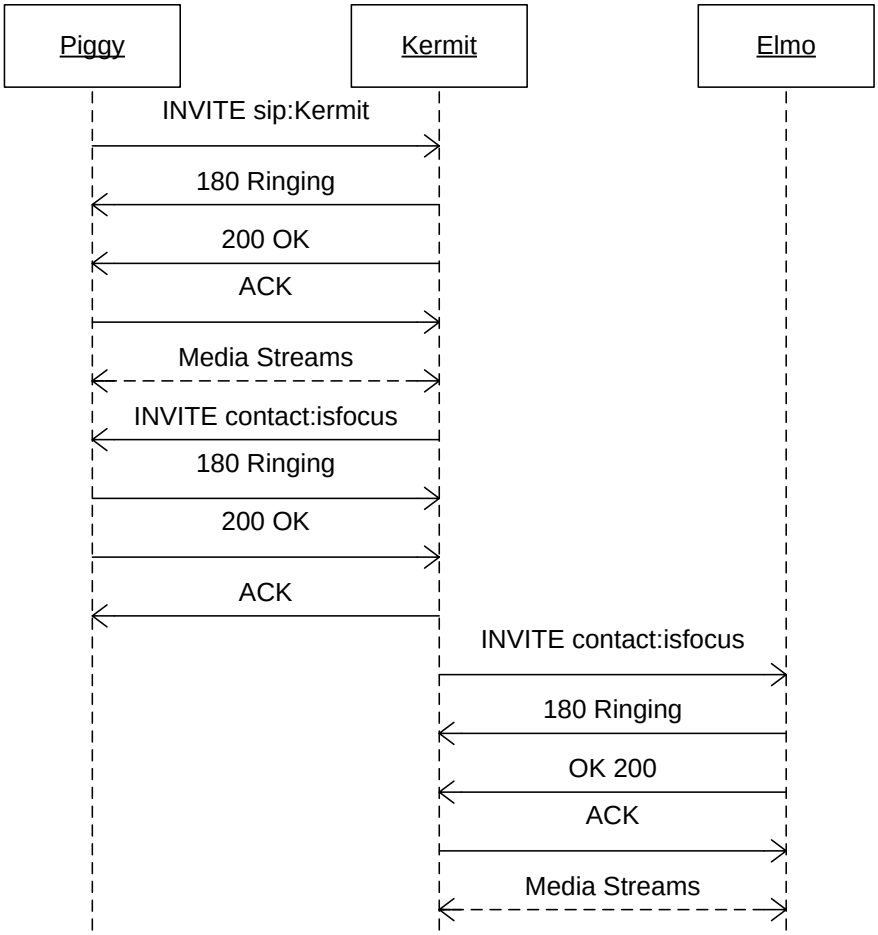

Figure 5: 3rd party is added

Like in the previous example a single conversation between the two user agents Elmo and Kermit already exits. Oscar, unaware that Kermit is in a call, invites him to create a new session. Kermit decides not to decline the call, but to initiate multiparty conversation. Therefore, he responds with the redirection class message 302 Moved including a newly generated conference URI and `isfocus` tag in the contact header. Such a SIP response could look like this:

```
SIP/2.0 302 Moved Temporarily
Call-ID: 0816@141.22.26.221
CSeq: 1 INVITE
From: "Oscar" <sip:oscar@sesamestreet.com>;tag=134652
To: "Kermit" <sip:kermit@sesamestreet.com>;tag=764684
Via: SIP/2.0/UDP 141.22.26.221:5060;branch=42
Via: SIP/2.0/UDP 141.22.26.117:5060;branch=23
Contact: <sip:puppets@sesamestreet.com>;isfocus
Expires: 3600
Content-Length: 0
```

Thereafter, Oscar sends a new INVITE to the conference URI and will be accepted

receiving the 200 OK response. Finally, Kermit re-invites Elmo to join the multiparty conversation. The call flow for this scenario is shown in figure 6. The advantage of this solution is the compatibility to every SIP stack implementing only the basic standard of RFC 3261 .

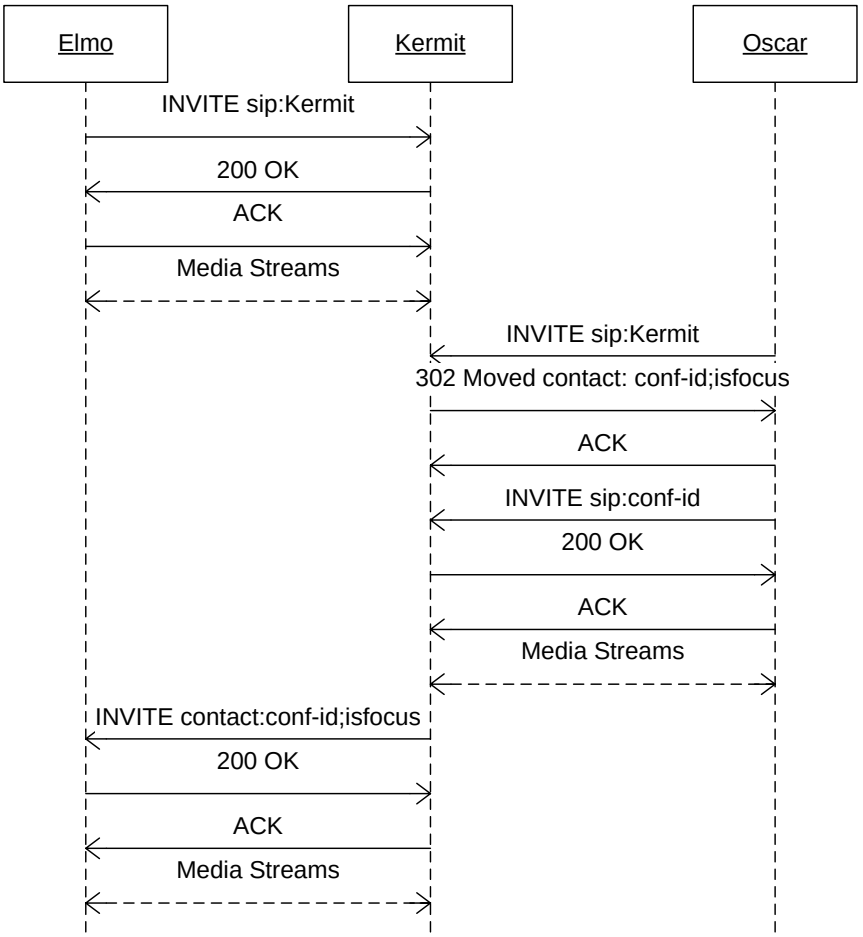

Figure 6: Third party initiated conference (alternative)

Such ad hoc created conferences can by joined by multiple participants like described in 2.2.1.

2.3 Approaches to Conferencing

This section presents related work about conferencing with the Session Initiation Protocol.

2.3.1 Centralized Conferencing

The idea of using a central point of control for multimedia conversations with SIP is a thoroughly discussed working area. In the Internet Engineering Task Force (IETF), especially two Working Groups (WG), the concluded Session Initiation Proposal Investigation WG (SIPPING) and the Centralized Conferencing WG (XCON), have established standards and frameworks for centralized conferencing. This section gives a short overview of the related work of both WGs. First, there have to be defined some high-level requirements for tightly coupled conferences with the Session Initiation Protocol [25].

- Discovery Phase:

  In a conference discovery phase, new participants must be enabled to locate an arbitrary SIP conferencing entity, and for a given SIP Address-of-Record (AOR), to resolve whether this entity have focus capabilities. It must be able to locate a conferences focus given a global identifier of the conference, and to obtain conference properties and state information. In cases, some of these requirements are may not fully implemented at the client side. For example in a ad hoc conference a participant may not subscribes for a notifications service retrieving conference information because it is unaware of the multiparty session.

- Conference Creation:

  As an entity with focus capabilities is available, operations must be defined for interested user agents to initiate an ad hoc or reserved conference identifier with specified properties. The difference between ad hoc and reserved identifier comprises from the expected duration for a specific conference. Reserved identifiers should be permanently accessible on a dedicated server in contradiction to ad hoc identifiers thats duration may as long as the conference is active.

- Participation of conference-unaware UA:

  A focus must be able to invite a SIP UA to a conference that is only compliant to minimal SIP implementations and likewise to accept a dial-in of such a UA. Such conference-unaware UA does not support for example the REFER method, Join header and ignores the `isfocus` tag, which are extensions to the SIP standard. This backwards compatibility assures that older implementations are able to participate in a conference.

- Dial-In, Dial-Out, Third-Party Invitation:

  Participation in a multiparty session can be achieved in three different ways. Dial-in, where the new member calls the focus, dial-out, where the focus invites the new participant, and third-party invite, where a conference member invites another UA to join a

multiparty by transmitting the conference identifier. The dial-in method is applicable for example to conference-unaware UAs described above. In a dial-out scenario, a participant may instructs a focus entity, to invite a list of user agents to the conference. To do so, it only sends one message to the focus instead of N messages for every party. That also remains compliant with standard SIP, because the focus will send INVITE requests. A third-party initiated invite goes the other way around. It request directly the expected user agent to enter into the conference. Because the required method REFER is no default implementation in SIP, the callee may decline the request and responds with a 501 NOT_IMPLEMENTED.

The SIPPING working group created a framework for conferencing with SIP and presented it in the RFC 4353 as informational work on how to establish a multiparty session with a protocol that normally is used for two-party sessions [35]. It shows an architectural model for conferences 3 and relies on the terminology already discuss above, e.g. focus, conference URI, conference policy etc. The requirements for tightly coupled conferences are realized in this framework. Required functions are distributed on different components that act as focus, as conference policy server, media mixers and as conference notifications server.

- Centralized Server:

  The most simple realization for a conferencing system is to place all component in one single physical server [35] including the media mixers as well. It can be foreseen that such "one box" solution does not scale well, because it has to handle signaling and media distribution simultaneous.

- Endpoint Server:

  This model aims at ad hoc and locally mixed conferences, after a two-party session changed to a multiparty session. The initializing user agent with focus capabilities serves as conference policy, conference notifications and media mixing server like in the centralized model above.

- Media Server Component:

  A more complex media server component model is shown in Figure 7. There are two servers for each conference. First, the application server that owns and manages the policies, memberships and maintains all dialogs with its participants. It appears as the central focus to the multiparty session, so that is handles all signaling. To perform the media mixing functions, it makes use of the second server called "mixer server". The mixer server implements a focus and a conference policy, as well, but its policy tells it to accept only invitations from the top-level application server. To inter-connect participants to the media streams, the application server uses third party call control

[37] for instructing the second server setting, thereby configuring at the mixers. If a participant sends a policy command to the focus in the application server, the focus delegates it to the media server by sending the same command.

This model has the advantage that the application server may controls more then one media server and counterwise, that a media server can be used for more than one conference application. The media servers are unaware of the conference policy and merely act as "slaves" to the application server. Note that for this model more than one physical machine can utilized. It is thus hardly applicable to ad hoc multimedia sessions.

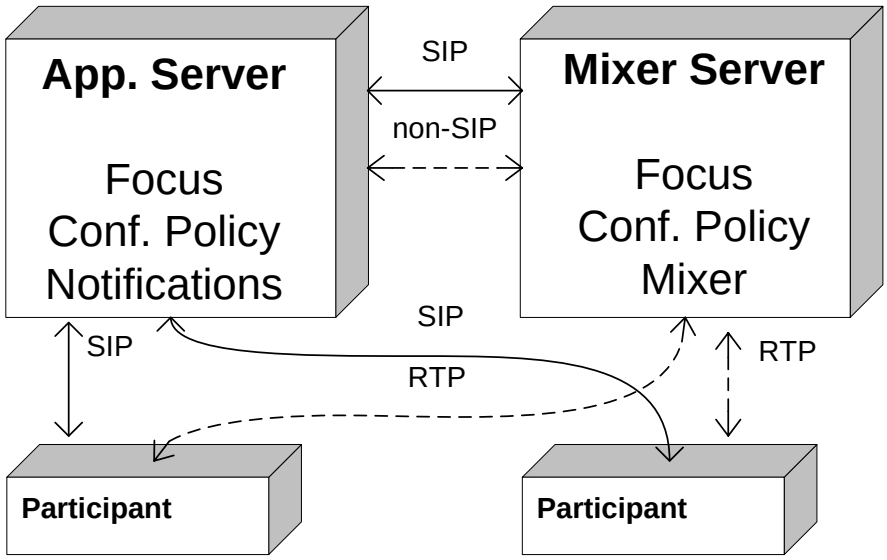

Figure 7: Media Server Component Model

- Distributed Mixing:

  The distributed mixing model is built on a delegation of media mixers. There is one endpoint that serves as conference server with focus, policy and notification capabilities. The media distribution is placed at conference members along with a conference policy as shown in figure 8. The media distribution is handled by the focus using third party call control and connects the media streams to each participants. Therefore, the focus holds a dialog with all participants to negotiate the sessions descriptions.

  The application of this model distributes the mixing among participants and may balance the load of every endpoint. However, it assumes that every UA in the multiparty session has enough hardware requirements to do so. In the case that an endpoint is a mobile device, this assumption may fail.

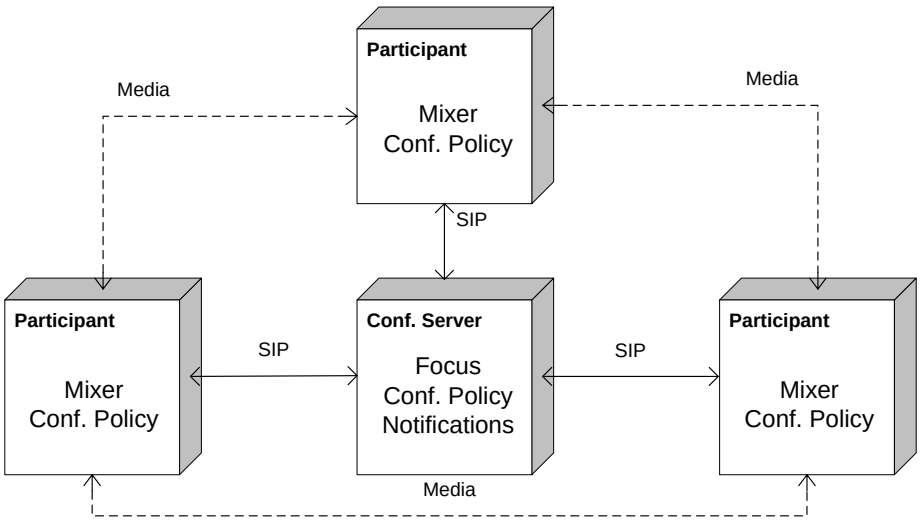

Figure 8: Distributed mixing model

- Cascaded Mixers:

  In a very large conference, it is impossible for one single media mixer to handle and distribute all media streams. In the architecture of cascaded mixers, there is one single focus endpoint that controls a set of mixers, typically implemented at multiple physical servers. The focus must use some control protocol to interconnect the mixers, so that all conference members receive the complete multimedia stream.

  In this model, an additional protocol is needed to assure the interoperability of the multiple media mixers. A large message overhead could be produced by synchronizing and controlling mixers with increasing size of the conference.

To standardize a notification service to the conference participants, the SIPPING WG created the "Event Package for Conference State" as defined in RFC 4575 [40]. It is build upon the "SIP-Specific Event Notification" [31] SIP extension. An UA can subscribe to the event package by sending a SIP SUBSCRIBE to conference URI. Notifications inform about changes of membership and optionally about changes in the conference state, like user counts or information about active users of a conference. Notification are performed by sending a SIP NOTIFY requests that carry an Extensible Markup Language (XML) content that updates the conference state at each endpoint. SDCON extends this event package for inter-focus communication and synchronization as described in section 4.

The Centralized Conferencing working group (XCON) at the IETF is developing a standardized suite of protocols for tightly-coupled multimedia conferences. They pay special attention to strong security and authorization requirements using the focus as central point of control as well. It picks up the central declarations of the SIPPING framework and adds additional services, e.g. floor control or sidebar conferences that may exist within the context of parent conferences. This framework offers richer functionalities, by including dedicated conference applications with explicitly defined capabilities located in the infrastructure. In most cases this stays in contradiction to ad hoc multiparty sessions.

2.3.2 Peer-to-Peer SIP

The dependency of traditional SIP on a proxy-registrar structure for message routing functions created the idea to replace this infrastructure by a peer-to-peer environment using a Distributed Hash Table (DHT) [6, 47] mechanism. The P2PSIP working group is developing a DHT based overlay network for REsource LOcation And Discovery (RELOAD) [21]. It specifies a couple of operations to join the overlay, putting resources into it (like the location of an UA) and fetching informations from the overlay network. In P2PSIP, there exists the possibility to add new services in the overlay like gateways or proxies to other networks with several protocols as shown in Figure 9. Each node named as "peer" in this figure is participating the P2P overlay network. They communicate between each other with a `P2P Peer Protocol` and are offering different services to the overlay. Peers can act as gateways to other networks or as proxy for standard SIP UA applications that are unaware to the usage of an overlay. The redirector peer will serve as an adapter towards incoming SIP calls from out the overlay to an P2PSIP-unaware user agent by responding the SIP URI of the requested callee. The "Client UA" on the right side, is not participating the overlay but can use its functions by using a `P2P Client Protocol`.

P2PSIP has not explicitly addressed the application area of group conferencing, but its adoption as "proxy and registrar" can be used to locate SIP UAs and to establish standard SIP dialogs and initiate conferences between them afterwards. Another advantage of using P2PSIP arises from its functionality supporting Network Address Translation (NAT) and firewall traversal. Many SIP endpoints are located behind NATs and firewalls that are hiding their IPs and blocking traffic. The P2PSIP WG is approaching this using protocol like Session Traversal Utilities for NAT (STUN) or Interactive Connectivity Establishment (ICE) to solve that issue.

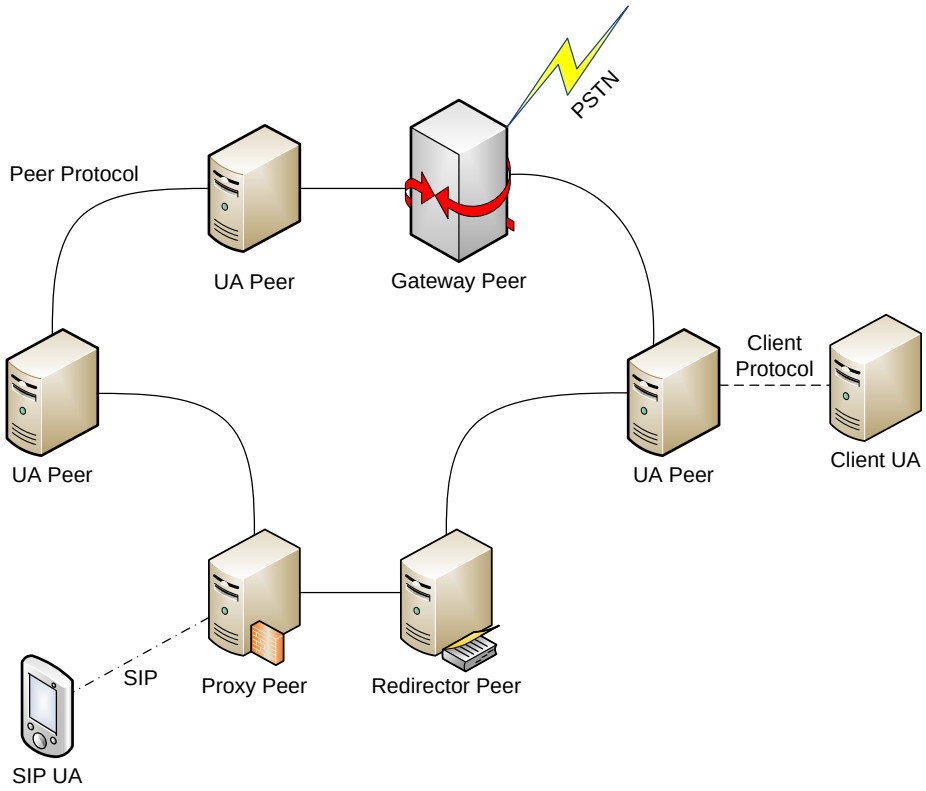

Figure 9: Overview of the P2PSIP Architecture

2.3.3 Related Work on Decentralized Conferencing

One example in the IETF for Distributed Conferencing (DCON) is published by a group from the University of Napoli [33, 32]. They define the requirements and created a framework for distributing Centralized Conferencing (XCON) [3] information on multiple XCON Servers. In order to do so, each XCON node is associated to a DCON module using a XCON-DCON Synchronization Protocol (XDSP) [34]. This tuple is called conferencing cloud like shown in figure 10. The DCON modules create an overlay network by interconnecting every conferencing cloud. The information shared within the overlay regards to currently active conferences provided at each conferencing cloud. An UA connected to an XCON application can retrieve all currently established multimedia session by using e.g., the Binary Floor Control Protocol (BFCP) [8]. The XCON server requests its DCON module to retrieve all conference states in the overlay and responds to the UA by giving it all conference information from the XCON and the DCON overlay. If the UA wishes to join a conference outside its own conferencing cloud (note, it is unaware of this fact) it sends a SIP INVITE to its XCON application. The XCON server is now responsible to place its client into the remote multiparty session, by performing

an `AddUser` operation to the DCON, that forwards the request to the remote conferencing cloud. If the remote XCON cloud accepts, it responds with an `UserAdded` operation. Finally the UAs XCON server issues a re-INVITE, revising the session descriptions. Following media streams are routed among the XCON servers to the client.

This DCON approach focuses on distributing conference information among dedicated conferencing servers and may not to be applicable to ad hoc conferences. To include a SIP UA into a remote conference, eight routing steps have to be performed. This seems to cause a long delay time, which is not desirable for conference applications. Performance measurements are not available at the time of elaboration.

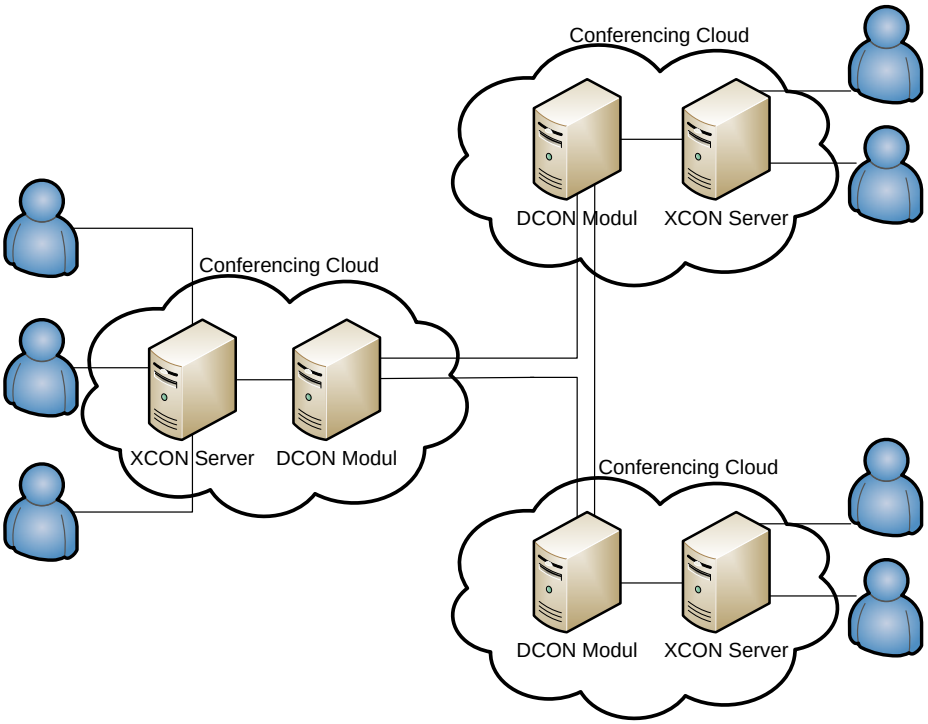

Figure 10: DCON architecture

A group from the Kyungpook National University in Korea designed a policy-based distributed management architecture for a large scale enterprise conferencing service using SIP [9]. In this approach dedicated conference applications including policy server, mixer management, notification service and focus capabilities can be divided on multiple physical servers in a hierarchical manner. They define two different kind of conference servers named by the focus functionalities "primary focus" and "regional focus". The primary focus represents the conference application in the way that its URI is used as the conference URI. Each participant must first obtain permission from the primary focus to enter a specific multiparty session. Each focus node controls a set of mixers. The primary focus is responsible to con-

figure and setup these mixers at different servers by creating a Conference Mixer Network (CMN) either in a star or tree topology. A regional focus is responsible for media mixing to its regional clients, that are assigned to it by the primary focus. The mixers of the regional focus are composed to a Local Mixer Network (LMN), distributing the media and taking care of participants assigned to it. A message flow displaying signaling and policy messages is shown in figure 11.

Like the DCON approach, this architecture needs an existing server infrastructure with dedicated conference applications. Measurements are displayed in the later section 6.

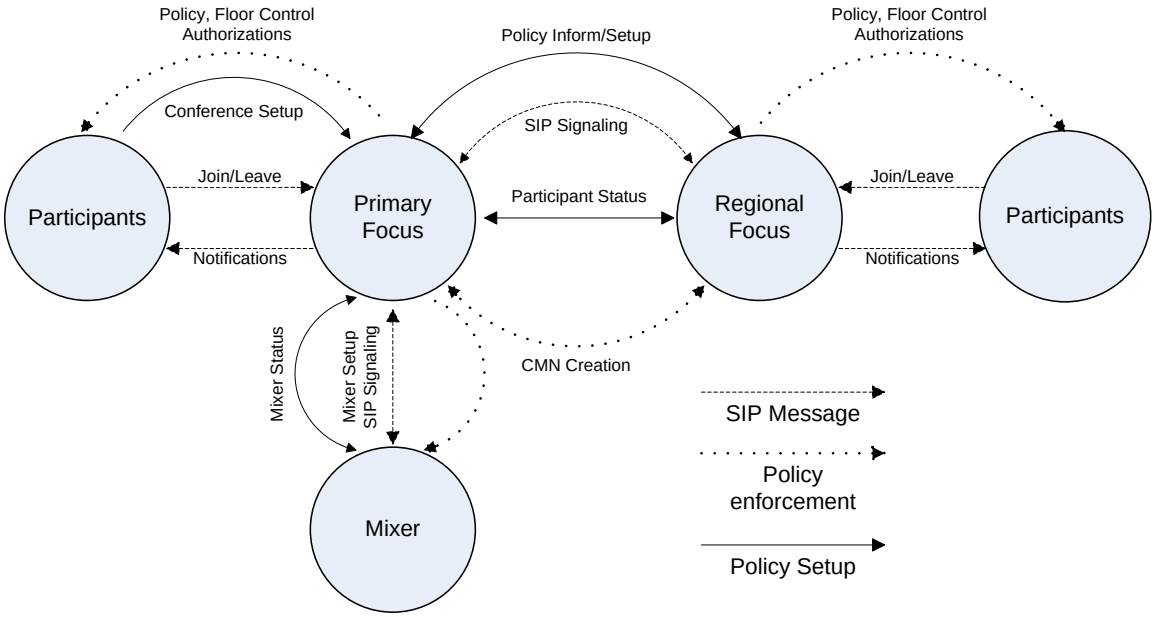

Figure 11: Message flow for the policy-based conference service as Cho et all [9]

3 Distributed Conference Control

This section presents the problem statement for a distributed conference control and a solution using the session initiation protocol.

3.1 Problem Statement

In a peer-to-peer scenario, a conferencing application faces the challenge to assure a stable communication platform among the peers, without assistance of a dedicated server infrastructure. In a case, where a single node becomes the central point of control and may simultaneous serves as multimedia distributor, its capabilities may be quickly exhausted. An example for this is a Skype video conference, which is limited to two participants. The central point of control concept in traditional tightly coupled conferences, implies the possibility of conference breakdown as the controlling entity leaves or crashes. To approach this problem, a distribution of the conference logic among multiple peers allows to compensate a departure of an controlling peer and sustain the multiparty service. Such a distributed solution can be adaptive of load peers must issue, to avoid that they will be requested to serve above their capacities. This dissociation of the focus role is not a standard behavior in SIP scenarios and needs some customization to be transparently conform and to SIP protocol operations. The distribution of an entity that still needs to appear as one location causes synchronization issues to keep a consistence view at all peers. The SDCON approach attempts to address these challenges in a SIP conformal manner, limiting extensions to the minimal.

3.2 A Concept for Scalable Distributed Conferencing

In an ad hoc environment, a two-party dialog that switches to a multiparty session forces one of the endpoints into the role of the focus of the conference. If willing, the focus can add or respectively accept new participants and increase the conference size. In SDCON we assume that a conference controller is aware of its own capacity limits for serving its client UAs. The participant which initiated the multiparty session is called "primary focus". When reaching its limits, the primary focus will require assistance by a so called "secondary focus". It will provide means to discover potential focus candidates among other conference parties. Such user agents need to implement the SDCON module, as well. A secondary focus is not functionally different to the primary focus. It will add, accept and manage calls from new participations and is able to discover other secondary focuses as well. Each focus is maintaining the set of SIP dialogs for participants directly connected to it. As shown in figure 12, each focus holds an SIP dialog to other focuses to facilitate status synchronization and state consistency. A specific topology of these inter-focus connections has purposefully not defined by SDCON, but is left to the individual conference instance. Participants are unaware whether they are connected to a primary focus or a secondary focus. Calls that

are addressed to the conference URI will normally routed to the primary focus. If this is fully booked, the primary focus will delegate these calls to remote secondary focus peers. Dial-out operations from conferences members to invite new participants will be routed to the focus whose they are connected to. The logical split of the focus role could be an alternative to the central point of control principle and the base for improved load balance in large-scale conferencing using SIP.

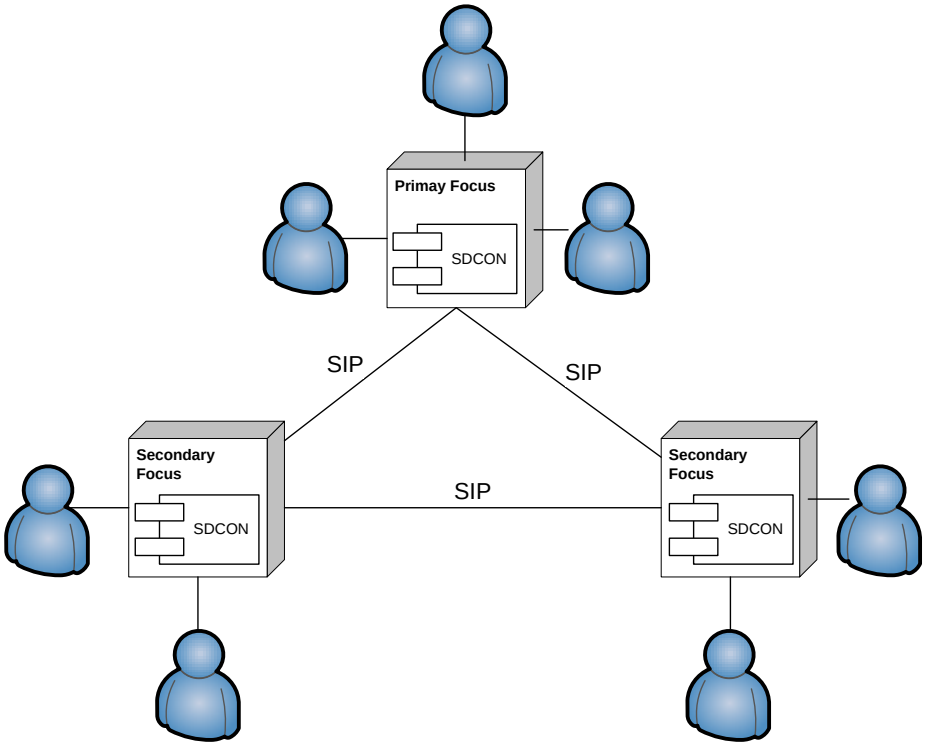

Figure 12: SDCON distributed session control

3.3 Challenges to SDCON

The concept of SDCON requires solutions to some challenges which will be described in the following section.

3.3.1 Globally Routable User Agent URI

Traditional solutions for tightly coupled conferences using SIP are commonly based on a central point of control that manages all conference functions and is addressable by a globally routable conference URI like explained the previous section 2.2. This URI identifies the

multiparty session and at the same time denotes the location of its focus. In SDCON, the functions of identifier and locator of a conference URI requires splitting, as there can be multiple points of control listening for requests under the same conference. Consequentially, the question arises on how can a SIP UA communicate with its corresponding focus entity (see figure 13) without a routing through the conference URI holder?

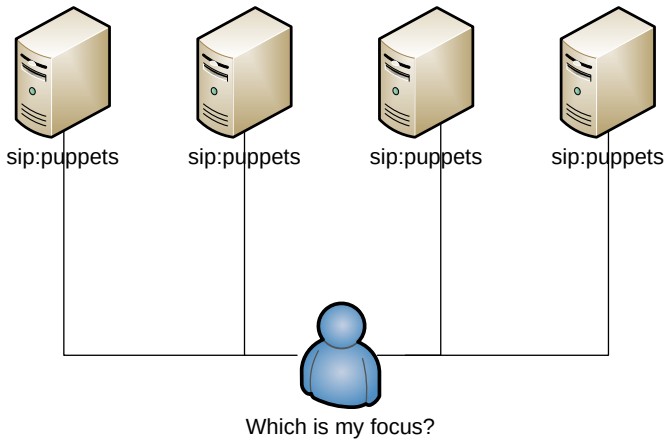

Figure 13: The multiple locator one identifier problem

3.3.2 Maintaining Consistence of Conference States

A distributed system is a combination of multiple independent computers that appear for a user as one coherent system [42]. To keep this fundamental requirement alive, there must be some kind of synchronization mechanisms between the various focus entities. Each focus entity adds or removes participants from the conference independently, so that the other focuses including the primary focus do not notice these changes. Without synchronization every focus peer would just know its own clients and remains blind with respect to the overall conference. This would smash the concept of a distributed conference control into multiple smaller centralized conferences without coherence.

It is also desirable to exchange information about the current state or capacities of the controlling peers i.e., the maximum size of accepting clients or references to other focuses for provide optimization proposes. In order to keep this synchronization simple, a SIP-based approach is desirable.

3.4 A SIP-based Approach

As described in the previews sections, the distribution of the conference controlling entity involves a coupled of problems that have to be fixed. In the following, this work presents

SIP-based techniques to distribute the focus entity among a couple of peers, transparent to the participant with common a consistent states.

An overall view of the message flows used in SDCON are shown in Figure 14. Between focus entities there are two main flows: the synchronization messages shown as "focus states" and the call delegation flows. Call delegations happen when a focus is fully booked and needs to refer additional calls to less loaded peers. Because calls to the conference URI will be routed to the primary focus, call delegations are mainly done from there to a secondary focus. Synchronization messages will be sent on every change of state in a single focus entity, e.g., announcing the arrival of a new participant. These messages have to reach every controller to keep a consistent view to the conference. Participants can join and leave a conference either by the primary or a secondary focus and may retrieve conference notifications by subscribing to these services. Another basic function consists in the ability to discover potential focuses with SDCON capabilities among peers intra-conference.

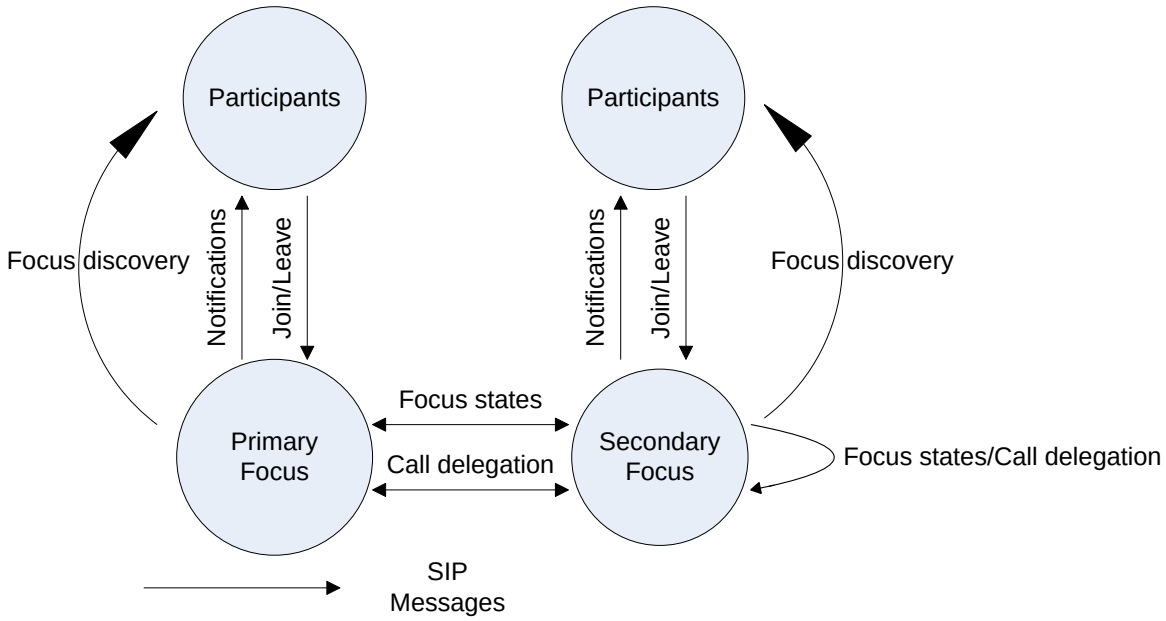

Figure 14: SDCON overall message flow

3.4.1 Discovery of an additional Focus

After a conference is created and the primary focus has reached its performance threshold indicated by the maximal number of members it can serve, the primary focus has to discover a new potential focus peer. A basic approach to detect another user agent which may operate as secondary focus is to consult the currently connected participants. As shown in figure

15, Kermit in the role of primary focus searches for a new secondary focus. He does so by sending a SIP SUBSCRIBE request suppliant for the `inet-multifocus-ci-ext+xml` event package embedded in the Event Header field. The name is composed by: "inet" for Internet Technologies, which is the group where this framework was developed, "multifocus", to indicate that the content is about multiple focus peers, "ci" for Conference Info, which is the base Event Package for Conference State [40], "ext" for Extension, because this package extends [40], "+xml", to indicate that this content is written in XML. To limit message overhead, the SUBSCRIBE request already carries a full XML conference state extension as described in section 4.

Such a SUBSCRIBE could be look like following:

```
SUBSCRIBE sip:oscar@sesamestreet.com SIP/2.0
Call-ID: 0817@141.22.26.55
CSeq: 1 SUBSCRIBE
From: "Kermit" <sip:kermit@sesamestreet.com>;tag=134653
To: "Oscar" <sip:oscar@sesamestreet.com>
...
Contact: <sip:Kermit@sesamestreet.de>
Event: inet-ci-multifocus-ext+xml
Expires: 3600
...
```

However, Kermit subscribes at Oscar for the event package extension to test for Oscar's SDCON abilities. In this example Oscar is not SDCON capable and responds with a non-affirmative 489 BAD EVENT response message. In this case, Kermit proceeds to the next peer in its SIP dialog list until a peer that supports the package will be found. If no suitable participant is around, incoming calls may be declined. Otherwise, a requested user agent responds with a 200 OK and thus becomes a secondary focus for this conference. Thereafter, Elmo subscribes to Kermit for the event package extension to retrieve event updates of on Kermit's state and to learn him about user capacities as encoded in XML format of the content body. In this way, any overloaded focus is able to redirect further join requests for the conference. A metric for the initiation of distributed conferencing can be individually defined by each focus peer and is beyond the scope of this work.

3.4.2 Call delegation

After one or more secondary focuses are discovered, the conference controlling peers are able to delegate incoming calls. Each focus peer can make the lookup operation `getDelegate()` on its local conference state information retrieving a potential remote focus willing to accept this call. An example for this behavior is displayed in figure 16. The conference-aware participant Oscar would like to make a dial-out invitation to the UA Ernie

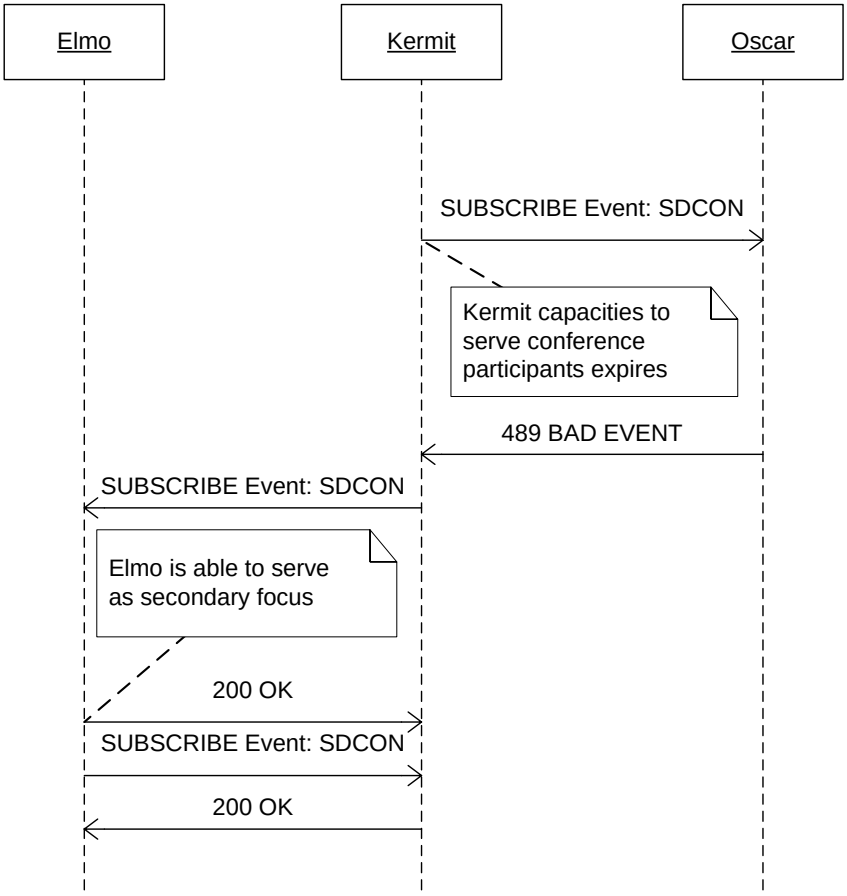

Figure 15: Discovery of a Secondary Focus

by sending a SIP REFER request to its focus. Kermit is fully booked and will not accept to manage this call, but he knows from his local conference state information, that the secondary focus Elmo is able to handle a new callee. Kermit then appears as it would accept the dial-out operation by responding an 202 ACCEPTED message and a notification that he is processing the REFER and forwards the call to Elmo. Elmo is now responsible for inviting the user agent declared in the SIP Refer-To header field. If successful, he notifies the referring focus and spreads a conference event to all focus peers which have subscribed to his notification service.

In another scenario shown in figure 17, the UA Grover has learned the conference URI by non-SIP means. Grover calls the primary focus Kermit via a SIP INVITE request. Kermit is fully occupied yet, but is aware that the remote secondary focus Elmo still has capacities

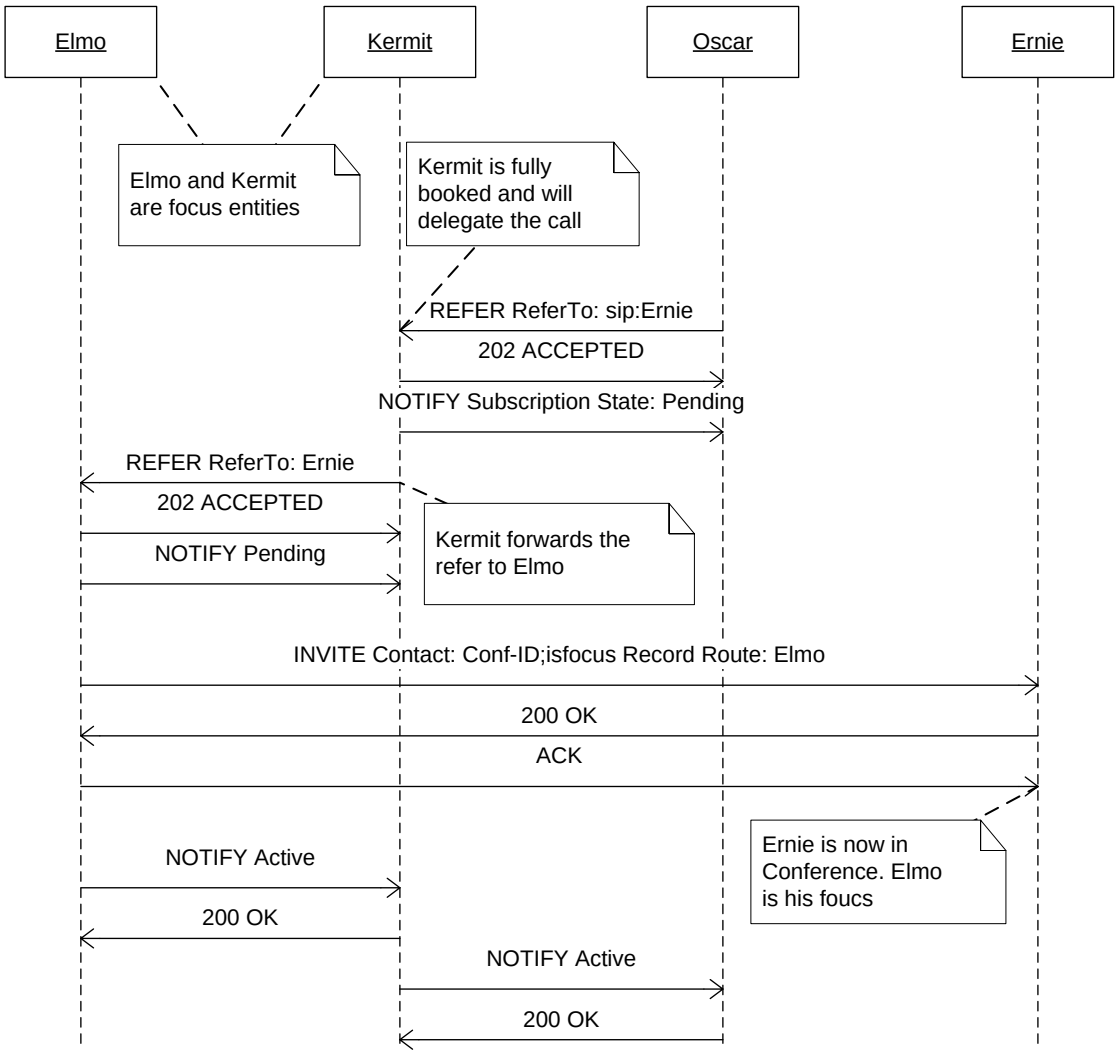

Figure 16: Delegation a call to a secondary focus

to accept calls. Kermit temporally accepts the request by responding 200 OK, but delegates it via SIP REFER to Elmo, who has now to re-INVITE Ernie to the conference.

3.4.3 Routing to a distributed Focus

In both call flows explained above, the INVITE operation issued by a secondary focus contains an additional Record-Route header field. The Record-Route header field is normally used as instruction for strict source routing by SIP proxies to force further requests in a dialog to be routed through the proxy [39]. This mechanism is used by SDCON to keep the appearance of a single logical focus peer and solves the globally routable user agent URI problem described in a previews section. Every secondary focus peer adds a Record-Route header with its own URI into every message sent to connected participants, e.g., the re-INVITE from Elmo to Ernie in figure 17 may look like following:

```
INVITE sip:ernie@sesamestreet.com SIP/2.0
Call-ID: 0818@141.22.26.55
CSeq: 1 INVITE
From: <sip:puppets@muppets.com>;tag=134652
To: "Ernie" <sip:ernie@sesamestreet.com>
...
Contact: <sip:puppets@muppets.com>;isfocus
Record-Route: <sip:elmo@sesamestreet.com>
...
```

The standard behavior of SIP implementations foresee that further requests within an existing SIP dialog must be sent through the "route set" containing a list of SIP URIs. It must be traversed when sending a particular request. A route set is a list of in reverse order that can be learned through headers like the Record-Route header field. If for example there is no other proxy between Elmo and Ernie, Ernie has to use the last entry of the Record-Route header field `sip:elmo@sesamestreet.com` as Request URI and the URI in the Contact header `sip:puppets@muppets.com` will be set as Route header. A Route header is used to force the next SIP entity to route the request to the URI taken of the Route header. Because Elmo is aware of the distributed focus mechanism and knows which UAs are connected to him, he will intercept every message addressed to its URI `sip:elmo@sesamestreet.com` and include the conference URI in the Route header and process the SIP request accordingly.

Requests initially sent to the conference URI will still be routed to the focus peer, that initiated the conference.

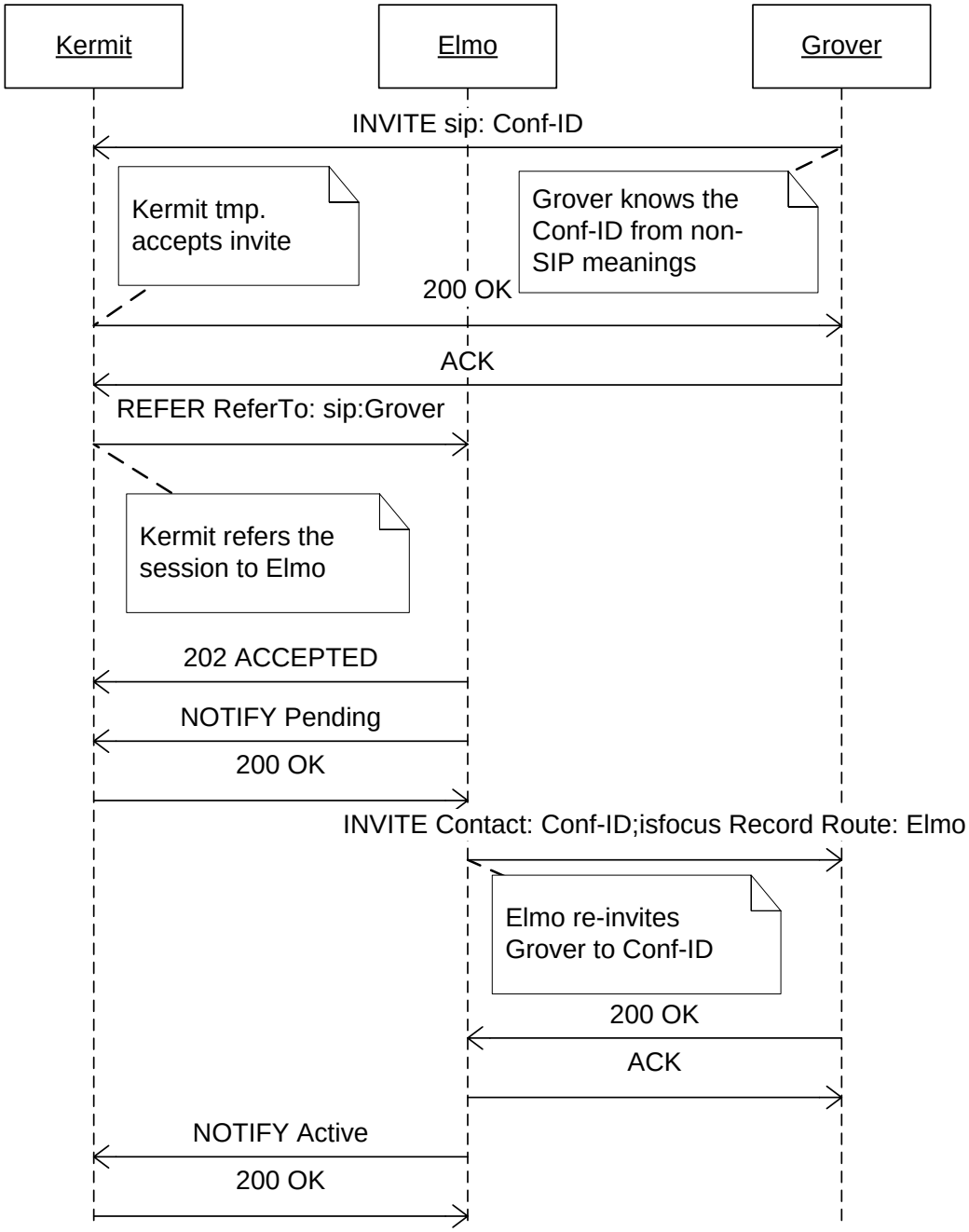

Figure 17: Call delegation in a dial-in scenario

3.4.4 Resilience Against Focus Failure & Leave

In contrast to centralized solutions, distributed conferencing operates without an single infrastructure component that guarantees a stable and always available management service. The scenario presented in this work targets at spontaneously created conferences between consumer computers or mobile devices. In such scenarios, leaves or failures of a focus device require fixing at low complexity.

At a graceful leave, the focus must delegate its established SIP dialogs to other, potentially available devices by sending SIP REFER requests for each of its peers. Subsequently, the leaving focus needs to terminate its event subscriptions by refreshing the SIP SUBSCRIBE requests with expiration headers set to zero. All focus entities are now aware of the focus departure and must no longer delegate SIP calls to it.

If a focus detects the failure of another focus, it could re-INVITE all conference participants connected to the absent controller or alternatively, try dial-out operations to other focus peers. These focuses then re-INVITE the participants to the conference again. Note that participants and their corresponding controllers are known from the distributed conference event package explained in chapter 4.

3.4.5 Consistency in a distributed Conference

As denoted in the previews sections, the local conference state information at every focus peers may change frequently on every join/leave operation or following newer focus discovery. The synchronization mechanism to keep a coherent and consistent state throughout the conference is done by subscriptions to the `inet-multifocus-ci-ext+xml` event state package extension. On every adoption of a new focus a bidirectional subscription will be established. Because SDCON does not force the conference controllers to arrange in a specific topology, a wild mesh of relationships between the various focus peers is possible. They may even arrange in a chain. In such a case, the focus states event information, i.e. the arrival of a new participant, has to traverse the complete chain of focus peers. This causes a long delay time to have a coherent state in the allover conference again. SDCON therefore just makes the requirement to avoid chaining by creating new subscriptions to remote focuses, which are in a "far" hop distance. A metric that define an adequate hop distance is not in scope of this work. The information whether a focus peer is "far" away can be obtained by the local multifocus conference state info.

The behavior of focus with SDCON capabilities at changes of states is simply to notify every other focus that is subscribed to the multifocus conference info event package.

The behavior of an SDCON focus on reception is as follows:

- Forward the notify to every focus subscribed to it, with the excepting of the originator
- Memorize which focuses have notify this peer, to prevent duplicate messages

Nevertheless, duplicate messages can not be complete prevented in arbitrary structures, but guarantee that every multiparty controller is informed about every change of state of the hole conference.

4 Consistent Conference State Information

This chapter introduces the use of the event package for conference states and presents an extension elaborated by this work.

4.1 Introduction

The SIP-Specific Event Notification [31] defines a subscription/notification mechanism for receiving events in SIP networks. These asynchronous notifications are useful for many SIP services as they enable a better communication between endpoints. Such services are, e.g., automatic call backs, buddy lists and presence informations or mailbox indications. All information distributed in this way are called 'Event Packages' anyone can be understood as an extension to the SIP-specific Event Notification. Event packages define a set of state information that the notifier reports to its subscribers. Subscriptions are soft states with a maximal duration that is defined in a SIP SUBSCRIBE request in the Expires header. To unsubscribe, a client sends a SUBSCRIBE request with expiration timer set to zero. To identify the event or class of events a user wants to be notified of, a SUBSCRIBE request must include exactly one Event header containing a token for identifying the subscription. A basic sample message flow of SIP-specific event notifications in shown in figure 18.

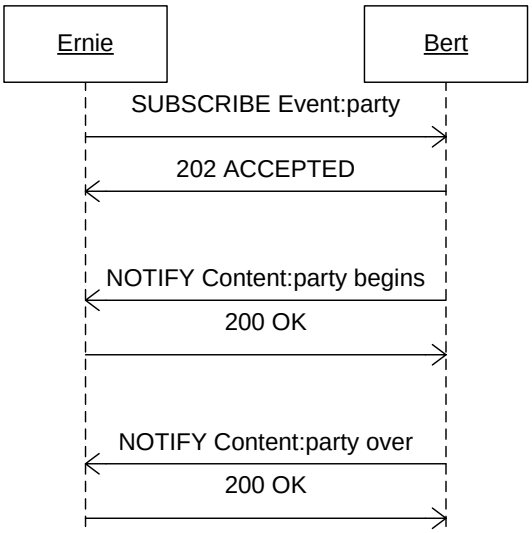

Figure 18: Subscribe/Notify call flow example

4.2 Event Package for Conference State

A conference focus in tightly coupled multiparty sessions is the only entity with sufficient information about the allover state and can offer a notification service to its participants. An event package has been defined by the SIPPING WG in A Session Initiation Protocol Event Package for Conference State [40].

The event package is defined in an XML schema and its name is registered at the Internet Assigned Number Authority (IANA) as `conference`. Its Multipurpose Internet Mail Extension (MIME) [15] type is `application` with subtype name `conference-info+xml`. An endpoint which wants to use this service subscribes for `conference` in the Event header field and must support the `application/conference-info+xml` data format that should be indicated in the Accept header field. If no such Accept header is present, it has the default value `application/conference-info+xml`.

Notifications are sent on every change of state in the conference and normally contain only the state that has change. This partial notification mechanism avoids redundant content replications and decrease the message sizes. Sub-elements from the XML root element `<conference-info>` use a 'state' attribute to indicate whether they are using this partial notification mechanism or not. Elements that are subject to frequently changes are permissible to attach the 'state' attribute with 3 the different values: 'full', 'partial' or 'delete'. An element with a 'full' state within a notification indicates that this element and all its child-elements have to be exchanged by this new state information. A 'partial' state only updates the state for a specific element and a 'delete' state forces to remove the complete content of the specific element. The following elements are permissible for partial notification mechanism:

- `<conference-info>`
- `<users>`
- `<user>`
- `<endpoint>`
- `<sidebars-by-val>`
- `<sidebars-by-ref>`

The conference XML schema provides a unique identification property among sub-elements of a common parent in a key-to-sub-element relationship. This makes it possible to use the partial notification mechanism as described above. A subscriber can discover which sub-element it has to update by using the key identifier. A replication of information in a document can also be prevented by using the key in a sub-element instead of repeating the hole

information again. Two type of keys are defined: 'entity' and 'id' of type `xs:anyURI` [46] in UTF-8 encoding. A list of elements using the key property is shown below:

- <user> with key 'entity'
- <endpoint> with key 'entity'
- <media> with key 'id'
- <entry> with key 'entity'

An example for the usage of keys and 'state' attribute could be look like this:

```
<conference-info>
...
  <!-- sub-elements -->
    <user entity="sip:oscar@sesamestreet.com" state="partial">
    ...
      <!--more sub-elements -->
    ...
    </user>
...
</conference-info>
```

A subscriber receiving this content has to change the data in its local document only for a user element identified by `sip:oscar@sesamestreet.com`.

An partial overview of the XML schema for the conference event package is shown in figure 19 with a short description of the elements:

- <conference-info>: Is the root element of the XML document.
- <conference-description>: A whole description of the multiparty like descriptive text or subject.
- <host-info>: Information about the entity hosting the conference.
- <conference-state>: This element gives an overview of the conference state like user count or if this multiparty is currently active.
- <users>/<user>: The `<users>` element serves as container for the `<user>` sub-element, each describing a single participant in the conference.
- <endpoint>: This element with parent-element `<user>` is used to provide signaling for sessions for a user's multiple devices.

- <media>: Information about a single media stream between the focus and the endpoint, e.g. the SDP negotiation.
- <sidebars-by-ref>/<sidebars-by-val> and <entry>: The event package foresees the possibility for cascading conferences inside a conference. The sidebar elements represent this feature using the `<entry>` element containing the sidebars conference URI. Subscribers for a conference state information may request to an additional subscription for a sidebar conference.

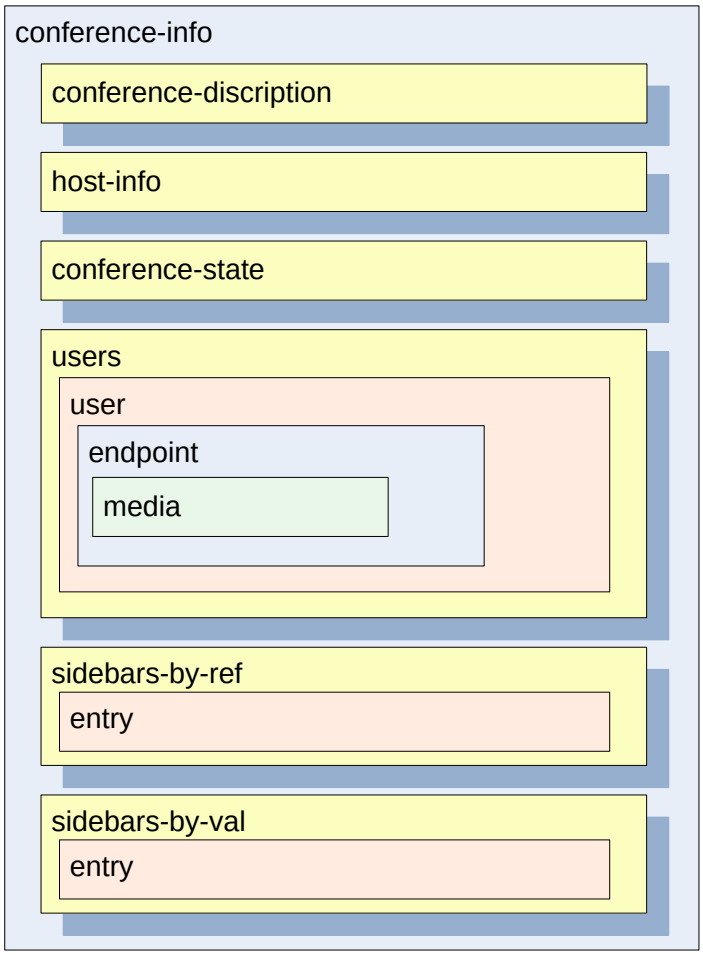

Figure 19: Conference event package overview

4.3 Multifocus Extensions

In contrast to traditional conferences in tightly coupled scenarios, the whole conference state information in SDCON is spread over the focus entities. This circumstance requires a synchronization among the focus peers.

The first step to approach this problem could be to distribute the conference event state information between the various focus peers. Therefore some semantics of the conference info XML event package needs to be changed in the application. In the case of a 'full state' reception for example, the new element should not override the locally established state, but rather should add this information to the available document. In this manner, all focuses envision at least a coherent state of the conference.

The challenge for a distributed conference control, however, lies in the knowledge about the splitting into multiple focus entities, in available background knowledge for load balancing, scalability enhancements and node leave/failure remediation. SDCON therefore provides in the second step corresponding information in the form of multifocus extension of the event package for conference states. By this extension, the distributed focus peers acquire an overview of the allocation of conference participants, their affiliation to their individual controllers, the willingness of a focus to service new participants and to display the established SIP subscription routes within the conference topology.

Figure 20 gives an overview of the additional multifocus conference information. An explanation for each element follows below:

- <focus-states>: This element is placed directly as child of the <conference-info> root element and serves as container for all following multifocus state information. Its XML complex type [45] definition is focus-states-type and it is permissible for the partial notification mechanism using the 'state' attribute. At the first subscription, it must carry a 'full' state information so that the corresponding endpoint can add the conference states information to its document. Thereafter, it is desirable to send only 'partial' focus states information on change.

- <focus>: Each `<focus>` element of type focus-type represent exactly one conference controlling peer to the conference. Because every focus in a SDCON environment is an participant, as well, this element uses the key 'entity' to reference to a `<user>` respective `<endpoint>` element from the base conference info document.

- <conf-id-holder>: This boolean element indicates whether this focus entity was the initial primary focus and is the owner of the conference URI.

- <focus-capacity>: This complex element has child elements `<max-participants>` and `<max-focus-references>`, both of type int, describes the capacities of this focus to avoid unsuccessful requests that could overload this peer. First, it informs

about the maximal number of participants it is willing to serve. Second, it indicates how many direct subscriptions of remote focus peers it will accept.

- <participant>: This element represents the list of SIP user agents which are connected to this specific focus peer. It uses the key attribute 'entity' to reference to a `<user>` respective `<endpoint>` element from the base conference info document and has no other child elements. Because every focus peer knows at which focus all participants are attached, this information can be used to reintegrate participants whose individual controller had left the conference by node leave or failure.

- <next-hops>: This element serves as container for the `<ref-to-focus>` of type `xs:anyURI` and displays to which other remote conference controllers this focus maintains a direct subscription for the conference event package extension. Using this information, each focus peer is able to comprehend the currently existing focus topology and to establish a new subscription on demand to avoid a 'far' hop distance to the other multiparty controlling peers.

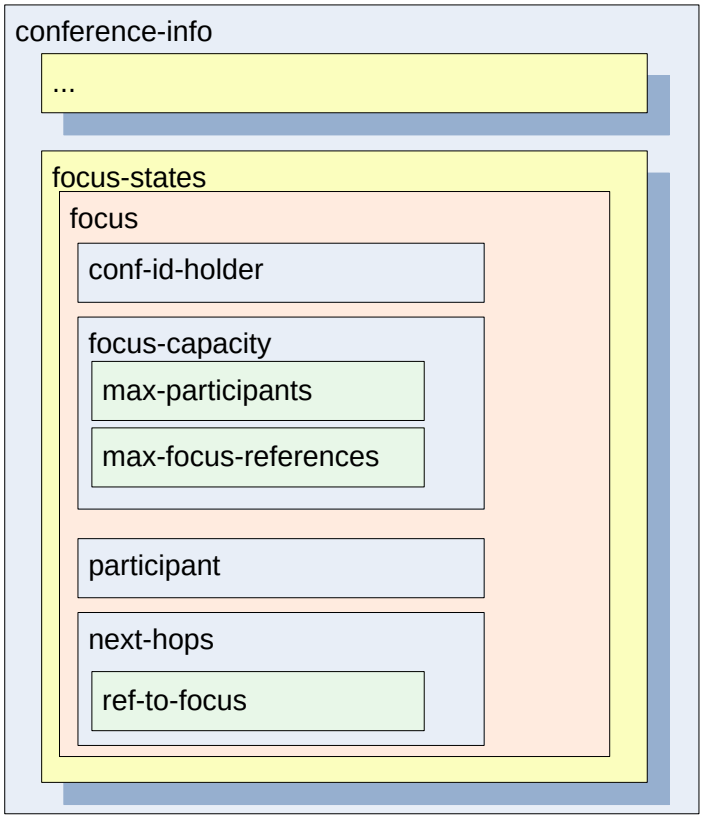

Figure 20: Extension for the conference event package

This extension to the conference event package is not built on the claim to be exhaustive and the XML schema annexed A at the end of this work foresees extendability for future requirements.

5 Implementation and Techniques

This chapter presents an overview of the implementation for the for distributed conference control framework designed in this work.

5.1 Language and Libraries

For the implementation of this framework, the programming language Java is used. The characteristic not to run native one the surrounding systems, but to be executed on the Java Virtual Machine (JVM) instead, permits the deployment on every operating system that has a virtual machine available. For this framework, the Java Development Kit 6 (JDK) and Java SE Runtime Environment version 1.6. is used and have been developed with the Eclipse IDE Version 3.4.1.

For testing and measurements, this work used the scripting language Python. It provides a comprehensive standard library, a simple and easy to understand syntax and is an object orientated programming language which uses an interpreter to execute a pre-compiled Python source code. An easy installation on various operating systems and the ability, that the same source code will run across these platforms without changes, makes it a useful tool for testing and measurements throughout this work. At state of elaboration, the Python version 2.6.2 has been used developed in Notepad++ text editor version 4.6.

5.1.1 JAIN-SIP

The Session Initiation Protocol Java APIs for Integrated Network (JAIN) work group solution is used in this framework as the base protocol implementation [19]. JAIN SIP is an open source standard-conformal Java interface to SIP for desktops and server applications [27]. It fully supports the protocol requirements for standard SIP implementations defined in [39] and includes some extensions, e.g., SIP-specific notification mechanisms or the SIP REFER method. JAIN SIP provides methods to format and parse SIP messages, send and receive messages, and provides an SIP transaction and dialog support. An overview of the JAIN SIP architecture is shown in figure 21 and a code snipped for stack instantiation follows below:

```
1 SipFactory sipFactory = SipFactory.getInstance();
2 sipFactory.setPathName("gov.nist");
3 Properties properties = new Properties();
4 properties.setProperty("javax.sip.STACK_NAME","Astack");
5 properties.setProperty("javax.sip.IP_ADDRESS","127.0.0.1");
6 SipStack sipStack = sipFactory.createSipStack(properties);
```

Listing 1: JAIN SIP Stack creation

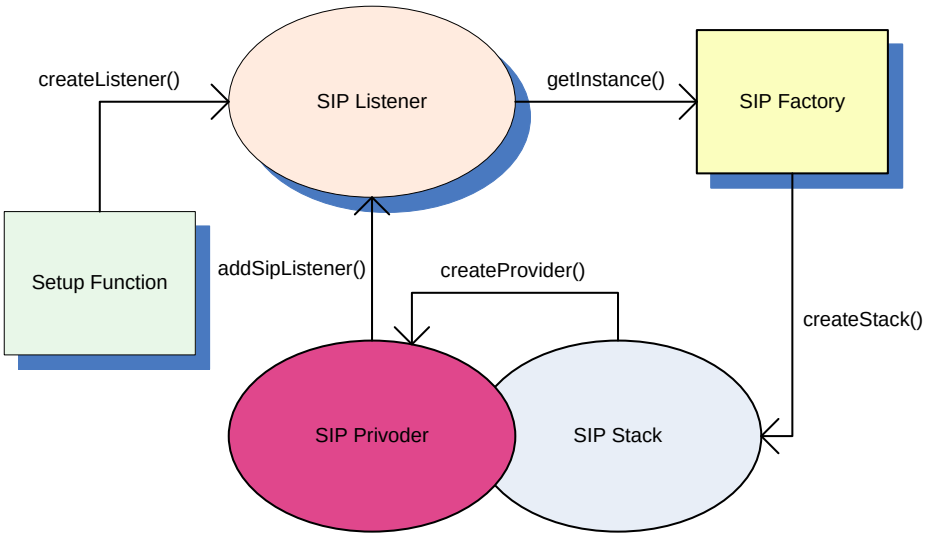

Figure 21: JAIN SIP architecture

The `SipStack` interface manages the listening endpoints and provides this associated with one single IP address. It is instantiated by the `SipFactory` passing stack properties via a `java.util.Properties` object. The `SipProvider` notifies the listeners about incoming requests and response messages, handles the SIP transactions statefully and creates call-Ids. The `SipListener` is the interface for the application logic and will be implemented by the developer.

In JAIN SIP, every message has to be created manually using header, address and messages factories prior to sending. This causes a high coding effort which makes the source code long , but has the advantage to allow for easy protocol modifications as they are used in SDCON. Its disadvantage derives from many unsupported SIP standard elements, e.g., no explicit support for the `isfocus` tag or the Join header is implemented. During our testing phase, it was figured out that this SIP stack sometimes did not follow the order of messages sending requests and elongated the implementation process.

5.1.2 JAXB

For creating and parsing XML documents as needed in the event package of conference state extensions presented earlier in this work, a Java Architecture for XML Binding (JAXB) reference implementation was used [20]. JAXB constitutes of a framework for processing XML document by providing a parser that creates a DOM tree of objects representing the content and structure of a given XML. Additionally, it can marshal a variety of objects into XML documents. For the latter task, the parser needs a list of classes for references of the Java object representations. With the XML Java Compiler (XJC), an XML schema can be

```
1  /*
2  * class instantiations and property set
3  * FocusStatesType.class is the generated by XJC
4  */
5  ctx = JAXBContext.newInstance(FocusStatesType.class);
6  marshaller = ctx.createMarshaller();
7  marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT,Boolean.TRUE); ↩
       marshaller.setProperty(Marshaller. ↩
     JAXB_NO_NAMESPACE_SCHEMA_LOCATION,SCHEMA_LOCATION);
8  unmarshaller = ctx.createUnmarshaller();
9  /*
10 * (Un)Marshalling from stream to Object, from Object to Stream
11 */
12 fst = (FocusStatesType)unmarshaller.unmarshal(xmlStream);
13 marshaller.marshal(fst, statesFile); // statesFile = XML document
```

Listing 2: Usage of JAXB

compiled into Java classes, where each XML element and XML type will produce one single Java Bean. The following code snippet shows its source code of its use.

Since Java version 6, developers have to take an eye on the XJC version when compiling the XML schema to get Java classes. The newer DOM parsers in Java standard library need 'annunciations' at Java Beans that defines which Java class is used as root element. Two solutions are possible, first, using a JAXB implementation version 2.0 or higher, second, add `@XmlRootElement` annunciations to the Java Bean representation of the root element. In this work, the second variate is used.

5.2 Software Design

The application implemented for this work is a hybrid of three functionalities. It can act as a simple SIP user agent initiating calls, refer other endpoints to a multiparty and request a notification service. If a secondary call arrives during an established two-point session, the application could make the decision to become the focus to this multiparty session and uses therefore its focus component as shown in figure 22. From now on, every signaling is handled by the focus component achieving the multiparty issues. If the conference remains in the limits preassigned in the 'capacity sheet', the application continues this mode. The capability sheet provides a coupled of values which will be queried by all application components to decide whether they have to change their executing mode. The focus component, e.g, queries on every incoming call the `MIN_NUM_FOR_DISTRIBUTION` value which is the indicator whether it has to create a multifocus environment. This value is $\leq$ to

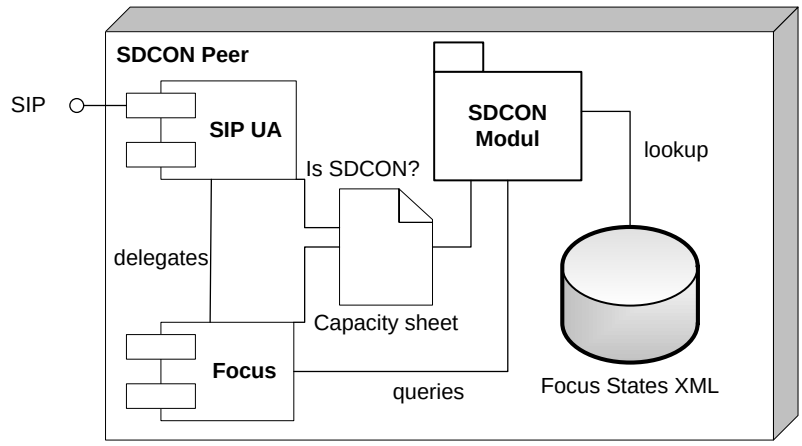

Figure 22: SDCON Peer components

the `MAX_PARTICIPANTS` value that defines the maximum number of participants this focus wants to service. On matching the `MIN_NUM_FOR_DISTRIBUTION`, the focus component will instantiate the SDCON module that immediately creates a multifocus states XML document using the values from the capacity sheet. The SDCON module has access to the list of SIP dialogs maintained from the focus and uses it to retrieve candidates to be requested for multifocus capabilities. Until reaching the `MAX_PARTICIPANTS` value, the primary focus peer still accepts calls and will notify its secondary focus about these events. On reaching the `MAX_PARTICIPANTS` value, all incoming calls be delegated to secondary focuses by making a lookup in the local focus states XML document to retrieve a less loaded node in the distributed environment.

A SDCON peer executing in SIP user agent mode will query its capability sheet on every incoming SIP SUBSCRIBE whether this request demands to change into multifocus mode. If true, the SIP user agent becomes a focus, will instantiate the SDCON module and uses the content of the subscription to create its focus states document. Thereafter, it generates a partial `<focus-states>` element including the values from the capacity sheet, adds this to its own states document and signals it to the originator focus peer via SIP SUBSCRIBE.

To represent this component architecture in source code, the SIP user agent component implements the JAIN SIP `SipListener` interface and registers at the SIP stack to retrieve SIP events. These events are thrown by the stack by incoming requests, responses, on dialog or transaction terminations and on I/O exceptions. When a the user agent has to convert to a focus, it instantiates a `SipFocus` object and delegates all further SIP messages to it. The `SipFocus` class also also implements the `SipListener` interface. Once, to have an similar source code structure for event handling, twice, because a user can choose start this application already in focus mode.

Displayed below in listing 3 is a sample method that has to be implemented by develop-

ers using `SipListener` interface. This method taken from the `SipClient` Java class, dispatches incoming SIP requests for further processing. Line 6 shows the delegation of messages when this peer act in focus mode. Line 11 displays the query that decides whether this peer has to change to a conference controller by looking up the capacity sheet.

```
1  @Override
2    public void processRequest(RequestEvent evt) {
3      Request req = evt.getRequest();
4      String method = req.getMethod();
5      ServerTransaction st = evt.getServerTransaction();
6      if(isFocus) { // all requests will be delegated to the focus
7        focus.processRequest(evt);
8        return;
9      }
10     if(method.equals(Request.INVITE)) {
11       if(CapacitySheet.makeConference()&&!isFocusRequest(evt)) {
12         processConferenceInitiation(evt);
13       } else {
14         processInvite(req, st);
15       }
16     }
17     else if(method.equals(Request.ACK)) {
18       processAck(req,st);
19     }
20     else if(method.equals(Request.NOTIFY)) {
21       processNotify(req,st);
22     }
23     else if(method.equals(Request.REFER)) {
24       processRefer(req,st);
25     }
26     else if(method.equals(Request.SUBSCRIBE)) {
27       processSubscribe(evt);
28     }
29   }
```

Listing 3: Request dispatching

Because every signaling between the focus peers is done via notifications and subscriptions, the focus component uses the SDCON module only for messages of these types. An example is shown in the following code snippet 4 out of the `SipFocus` implementation. On receiving a SIP NOTIFY the SDCON module `disFocus` will be ask whether this request belongs to multifocus issues, and if true, to handle these requests like shown in lines 4,6 and 9.

```
1  else if(method.equals(Request.NOTIFY)) {
2    SubscriptionStateHeader substate =(SubscriptionStateHeader)req. ↩
       getHeader(SubscriptionStateHeader.NAME);
3      if(disFocus != null) { // a distributed conference?
4        if(disFocus.isConferenceRequest(req)) {
5          if(substate.getState().equals(SubscriptionState.PENDING)) {
6            disFocus.sendOK(req, st);
7          }
8          else if(substate.getState().equals(SubscriptionState.ACTIVE) ↩
             ) {
9            disFocus.processConfRequest(req, st);
10         }
11       }
12       else {
13         /* other case */
14       }
15     }
16 }
```

Listing 4: Processing notifications

A lookup in the focus states XML document by the SDCON module, retrieves the SIP URI of a less loaded focus peer. The dialog management service class `Sessions` then returns the corresponding SIP dialog to this URI as shown in the next code snippet 5. An incoming request that would overload this peer will be delegated to the elected remote focus. A `Dialog` object includes a data structure carrying all necessary information to create a new SIP request and delegating the call.

```
1 Dialog delegatee = Sessions.getSDCONDialog(FocusStates.getDelegate());
```

Listing 5: Focus States lookup operation

5.3 Implementation Progress

This section presents relevant excerpts of the implementation for distributed conference control.

5.3.1 Call Delegation

Within call delegation, there exit two different variants that have to be considered at the implementation level. First, the delegating focus peer has a subscription dialog to the remote focus, or, second, it knows the remote peer only from its local focus states document. This

decision and the response operation for accepting a REFER request are shown in listing 6. This methods is used at the SDCON module which have obtained the `Request` and the `ServerTransaction` objects from focus component. The method extracts the conference URI from the request and afterwards, checks if this request was of type REFER respectively INVITE. A REFER requests needs to be responded by 202 ACCEPTED message and needs notification indicating, that this request is pending. Because request events will be thrown asynchronous from the SIP stack, the application has to store the request and transaction objects for later on usage. The statement `FocusStates.getDelegate()` makes a lookup for a potential focus peer like explained in an anterior section. The Session service class `Session` then tries to get a saved dialog for the result SIP URI from the lookup. If successful, the delegation can be done within an existing subscription dialog, otherwise a complete new request has to be created.

```
1  public void delegateToNext(Request req, ServerTransaction st) {
2      try {
3        SipURI conferenceURI = (SipURI)((ToHeader)req.getHeader(ToHeader ↩
             .NAME)).getAddress().getURI();
4        if(st == null)
5          st = sipProvider.getNewServerTransaction(req);
6        // ACCEPT and Notify refer Request
7        if(req.getMethod().equals(Request.REFER))
8          acceptedNotify(req, st);
9        // store original Request and Transaction
10       originalRequestQueue.offer(req);
11       originalServTransQueue.offer(st);
12       Dialog subscribeDialog = Sessions.getDCONDialog(FocusStates. ↩
             getDelegate());
13       if(subscribeDialog != null) {
14         Address conferenceAddress = addressFactory.createAddress( ↩
               conferenceURI);
15         delegateInnerDialog(req, st, subscribeDialog,  ↩
               conferenceAddress);
16       }
17       else { // I know the Focus but got no SDCON Dialog to him
18         delegateOuterDialog(req, st, FocusStates.getDelegate(),  ↩
               conferenceURI);
19       }
20     } catch (Exception e) {
21       e.printStackTrace();
22     }
23   }
```

Listing 6: Delegation dispatching

Listing 7 shows the creation of a REFER request, that will delegate a call to a remote focus peer and takes an example how a request is build in JAIN SIP with an existing dialog object. This code snipped could be executed as a direct result from the previews example 6 as the `Session` service object returned a `Dialog` object for the SIP URI obtained by the lookup operation. To create a new REFER message, the application again has to ascertain whether the original request was a dial-in operation or if a dial-out was foreseen to set the correct SIP URI into the Refer-To header field. The statement `subscribeDialog.createRequest(Request.REFER);` creates a new request using the existing `Dialog` object. Only the Refer-To header field as to be added manually to accomplish the new request sent the remote focus peer. Finally, the local data structure for this subscription dialog has to be updated as shown in line 16.

```
1  private void delegateInnerDialog(Request req, ServerTransaction st,
       Dialog subscribeDialog, Address conferenceAddress) {
2    try {
3      // BEGIN a new Refer
4      ToHeader refToHeader = headerFactory.createToHeader(
         conferenceAddress, subscribeDialog.getRemoteTag());
5      ReferToHeader referToHeader = null;
6      if(req.getMethod().equals(Request.REFER))
7        referToHeader = (ReferToHeader) req.getHeader(ReferToHeader.
           NAME);
8      else if(req.getMethod().equals(Request.INVITE))
9        referToHeader = headerFactory.createReferToHeader(((FromHeader
           )req.getHeader(FromHeader.NAME)).getAddress());
10     ContactHeader refContactHeader = headerFactory.
         createContactHeader(hostNameAddress);
11     Request forwardsRefer = subscribeDialog.createRequest(Request.
         REFER);
12     forwardsRefer.addHeader(referToHeader);
13     // END creating new Refer
14     ClientTransaction ct = sipProvider.getNewClientTransaction(
         forwardsRefer);
15     ct.sendRequest();
16     Sessions.updateSDCONDialog(subscribeDialog);
17   } catch (Exception e) {
18     e.printStackTrace();
19   }
20 }
```

Listing 7: Delegation to known focus

5.3.2 Creating Focus States

The representation for XML documents in Java Bean objects gain more importance in the Java community. In the areas of Interface Definition Language (IDL) and especially for Web services often a source code representation in a neutral format is eligible. From out the second area a coupled of libraries have been developed to serve this market. The sample code snippet shown in listing 8 shows the usage of Java Beans created by the XJC tool from the JAXB reference implementation. This method taken from the `FocusStates` service object to creates a `FocusStatesType` object that reflects the `<focus-states>` root element from the conference state extension. Because this object has Java Bean character, it uses setter and getter methods to handle its values. For sub-elements that are defined for a cardinality `0..*`, there exists no setter method. To add new elements to such an element, the getter methods returns a list implementing `java.util.List` which provides its own `add()` method like shown in line 24.

```
1  private static FocusStatesType createFocusStatesType(StateType state, ↩
       SipURI focusEntity,
2        Integer maxParti, Integer maxRef, SipURI[] participant,SipURI  ↩
           refTo) throws XMLStateException {
3      FocusStatesType fst = new FocusStatesType();
4      fst.setState(state);
5      if (focusEntity == null)
6        throw new XMLStateException("Partial state makes no sense  ↩
             without FocusType");
7      FocusType ft = new FocusType();
8      ft.setEntity(focusEntity.toString());
9      ft.setState(state);
10     if (maxParti != null || maxRef != null) {
11       FocusCapacityType fct = new FocusCapacityType();
12       fct.setState(state);
13       if (maxParti != null)
14         fct.setMaxParticipants(maxParti);
15       if (maxRef != null)
16         fct.setMaxFocusReferences(maxRef);
17       ft.setFocusCapacity(fct);
18     }
19     if (participant != null) {
20       for (int i = 0; i < participant.length; i++) {
21         ParticipantsType pt = new ParticipantsType();
22         pt.setEntity(participant[i].toString());
23         pt.setState(state);
24         ft.getParticipants().add(pt);
25       }
```

```
26      }
27      if (refTo != null ) {
28          GraphType gt = new GraphType();
29          gt.setRefToFocus(refTo.toString());
30          gt.setState(state);
31          ft.getGraph().add(gt);
32      }
33      fst.getFocus().add(ft);
34      return fst;
35  }
```

Listing 8: Creating of a focus states document

5.3.3 Updating Focus States

An imported aspect for distributed conferencing is to provide a consistent state to the overall conference in the local focus states document at each peer. To achieve this issue on receiving an `inet-multifocus-ci-ext+xml` event, the method `setPartial()` from the `FocusStates` service class, shown in listing 9, is responsible. In the simplest case, a new focus states information is of type 'full'. It then can override the locally state information by the newer one. If not, this method iterates over each element and its sub-elements comparing it with the received states information and updates it on detecting differences. If an element is of state 'delete' it must remove this element for the document. Finally, the the focus state is updated and a file copy will be created like shown in line 74.

```
1   public static void setPartial(FocusStatesType partial) {
2     boolean isEstablishedFocus = false;
3     if (partial.getState() == StateType.FULL) {
4       try {
5         setFullStates(partial);
6       } catch (XMLStateException e) {
7         e.printStackTrace();
8       }
9       return;
10    }
11    FocusStatesType localStats = (FocusStatesType)partial.clone();
12    FocusType newFocusState = localStates.getFocus().get(0);
13    newFocusState.setState(StateType.FULL);
14    if (focusStates != null) {
15      for (int i = 0; i < focusStates.getFocus().size(); i++) {
16        if (newFocusState.getEntity().equals(
17            focusStates.getFocus().get(i).getEntity())) {
```

```
18          isEstablishedFocus = true;
19          FocusType oldFocusState = focusStates.getFocus().get(i);
20          // BEGIN ConfIDHolder Set
21          oldFocusState.setConfIdHolder(newFocusState
22              .isConfIdHolder());
23          // END ConfIDHolder Set
24          if (newFocusState.getFocusCapacity() != null) {
25            oldFocusState.setFocusCapacity(newFocusState
26                .getFocusCapacity());
27          } // END IF Capacity check
28          if (newFocusState.getGraph() != null) {
29            for (GraphType newGraph : newFocusState.getGraph()) {
30              if (newGraph.getState().equals(StateType.DELETED)) {
31                List<GraphType> oldGraphs = oldFocusState.getGraph();
32                for (int j = 0; j <= oldGraphs.size(); j++) {
33                  if (oldGraphs.get(j).getRefToFocus().
34                      equals(newGraph.getRefToFocus())){
35                    oldFocusState.getGraph().remove(j);
36                  }
37                }
38              } else {
39                if(!oldFocusState.getGraph().contains(newGraph) &&
40                   newGraph.getRefToFocus() != null) {
41                  newGraph.setState(StateType.FULL);
42                  oldFocusState.getGraph().add(newGraph);
43                }
44              }
45            }
46          } // END IF Graph check
47          if (newFocusState.getParticipants() != null) {
48            for (ParticipantsType newParticipant : newFocusState. ↩
                getParticipants()){
49              if (newParticipant.getState().equals(
50                  StateType.DELETED)) {
51                List<ParticipantsType> oldParticipants = oldFocusState
52                    .getParticipants();
53                for (int j = 0; j < oldParticipants.size(); j++) {
54                  if (oldParticipants.get(j).getEntity()
55                      .equals(newParticipant.getEntity())) {
56                    oldParticipants.remove(j);
57                  }
58                }
59              } else {
```

```
60                if (!oldFocusState.getParticipants().contains( ↩
                      newParticipant)) {
61                  newParticipant.setState(StateType.FULL);
62                  oldFocusState.getParticipants().add(newParticipant);
63                }
64              }
65            }
66          } // END IF Participants check
67        } // END IF Entity Check
68      } // END FOR Focus select
69      if (!isEstablishedFocus) {
70        focusStates.getFocus().add(newFocusState);
71      }
72    } // END IF focus states
73    try {
74      marshaller.marshal(focusStates, statesFile);
75    } catch (Exception e) {
76      e.printStackTrace();
77    }
78    sortFocusStates();
79  } // END METHOD setPartial
```

Listing 9: Updating the focus states

5.3.4 Application appearance

At last, a simple graphical user interface (GUI) has been developed for testing, debugging and representative purposes 23. It supports all functionalities shown in the previous section and has been created with the Java widget toolkit Swing which makes it easy portable to other systems without an installation.

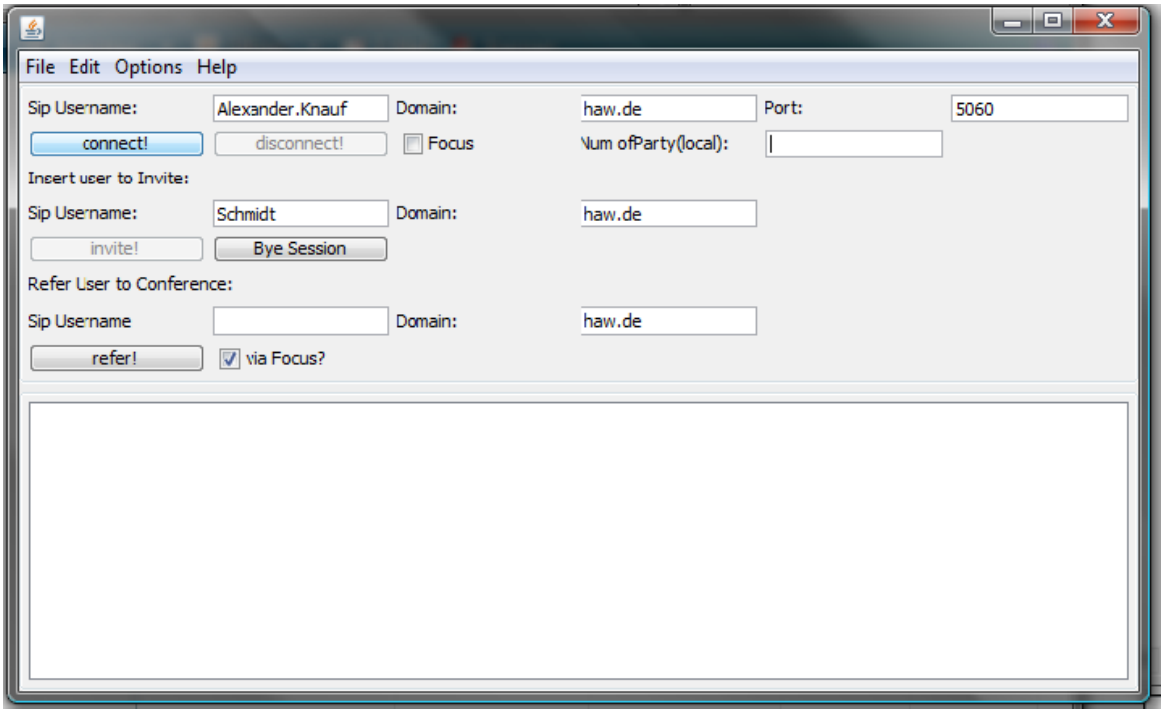

Figure 23: SDCON Peer GUI

6 Measurements and Evaluation

This chapter demonstrates the attained functionalities as a proof of concept and presents two measurements that display the base performance and scalability for distributed conference control.

6.1 Dial-In Operation

6.1.1 Measurement Setup

The first presented measurement analyzes the signaling delay for callers willing to participate in a conference. This dial-in scenario operates by sending a SIP INVITE requests to the conference URI. These measurements were performed in evaluation mode using three physical endpoints that host multiple SDCON applications, each emulating a single participant by using distinct ports. The schematic measurement setup is displayed in figure 24. The primary focus node and a couple of participants connected to it are placed on a single system. A number of secondary focus nodes and clients connected to them are placed on other devices. In the real scenario, it have been instantiated 51 SDCON nodes divided onto six peers running in focus mode and 45 executed as standard SIP UA. Every focus node was

parametrized to accept 10 incoming calls and to initiate a focus discovery operation after accepting the 9th call.

The topology shown in figure 24 reflects the organic build-up during measurements. The first two calls where sent to the user agent's URI which then changes to focus execution mode by creating the conference URI, and re-inviting the previously accepted user agent to the multiparty session. Thereafter, every new instantiated UA calls the conference URI which will be routed to the primary focus. As it reaches the `MIN_NUM_FOR_DISTRIBUTION` threshold, it initiates a focus discovery among the currently connected participants. After accepting another incoming call, it reached its predefined limit of service calls and delegates all further calls to the secondary focus by sending SIP REFER requests as explained in section 3.4.2. As soon as the secondary focus also reaches its maximum capacity, it likewise performs a focus discovery. At this point, there are three controlling nodes, two of them fully booked, and one willing to service 10 more participants. Note that the focus peers are aware of this conference status by exchanging the synchronization messages displayed in figure 24 by the dashed lines. Consequently, further requests received at the conference URI will be delegated directly to the recent discovered focus with free capacities. To avoid long routes and synchronization overhead, the primary focus subscribes to the multifocus conference event package extension at the recently discovered secondary focus. In this scenario, a star topology for call delegations and a composition of ring and star topology for synchronization messages is built.

6.1.2 Measurement Results

The measurement results displayed in table 1 represent the time for a single participant to establish a SIP dialog with a controlling entity. The first column presents the measurements for the distributed conferencing scenario as described above. The next two columns are measurements extracted from the work of Cho et all [9]. The 'Centralized' column shows the signaling delay times for traditional tightly coupled conferences using a single point of control. The 'Hierarchical' column represent the measurements for the signaling delay in the policy-based distributed management architecture for a large scale enterprise conferencing elaborated by Cho et all.

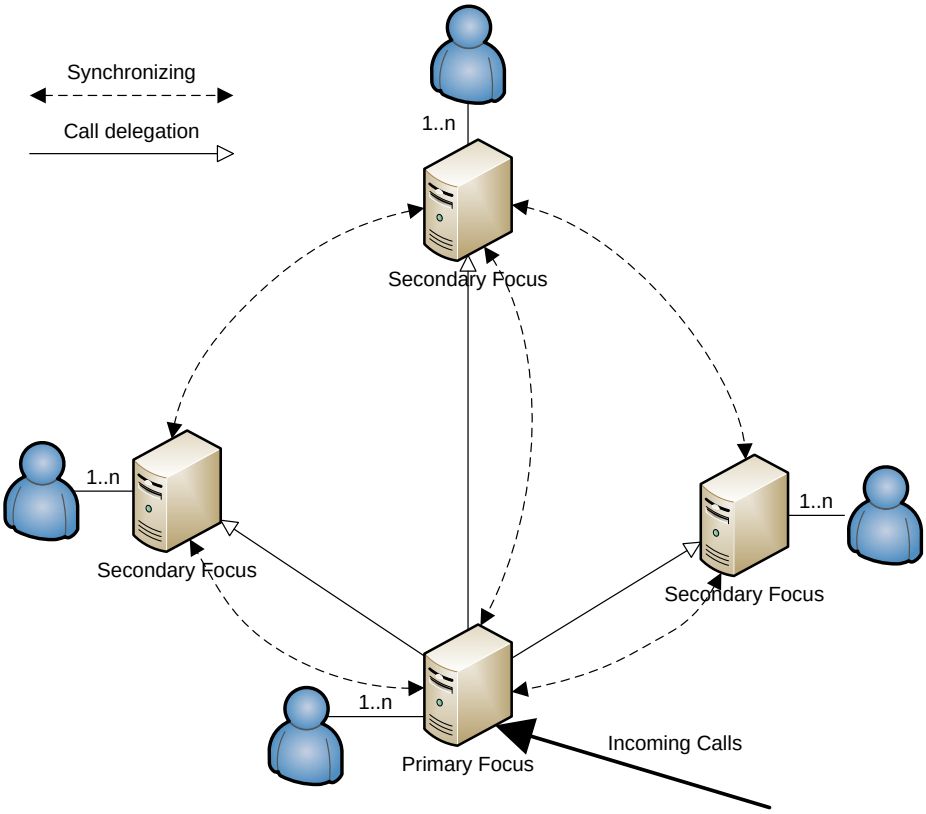

Figure 24: Calling the conference URI

Because the conditions in terms of absolute performance delays, i.e., implementation environment, processing time, network latencies etc., for both measurements were not the same, the SDCON measurement results have been normalized to the Cho at all ratings. The centralized measurement values were used as reference numbers for normalizing the absolute end-to-end delays. A SDCON focus serves as central point of control, like in the centralized architecture, until it reaches its limits. All SDCON absolute numbers are correspondingly scaled by a coefficient

$$c = \frac{\sum_{n=0}^{n} \frac{x_{ci}}{x_{di}}}{n}$$

$$c = 1,135$$

to achieve a realistic comparison. The SDCON emulation measurement results are showing the average delay time obtained from 20 independent runs.

Participants #	SDCON /ms	Centralized/ms	Hierarchical/ms
1	53	51	110
5	51	50	108
10	53	51	110
15	90	50	108
20	91	52	110
25	95	50	108
30	95	50	120
35	94	50	123
40	93	51	130
45	85	50	160
50	93	52	175

Table 1: Measurement results for SIP dialog establishment

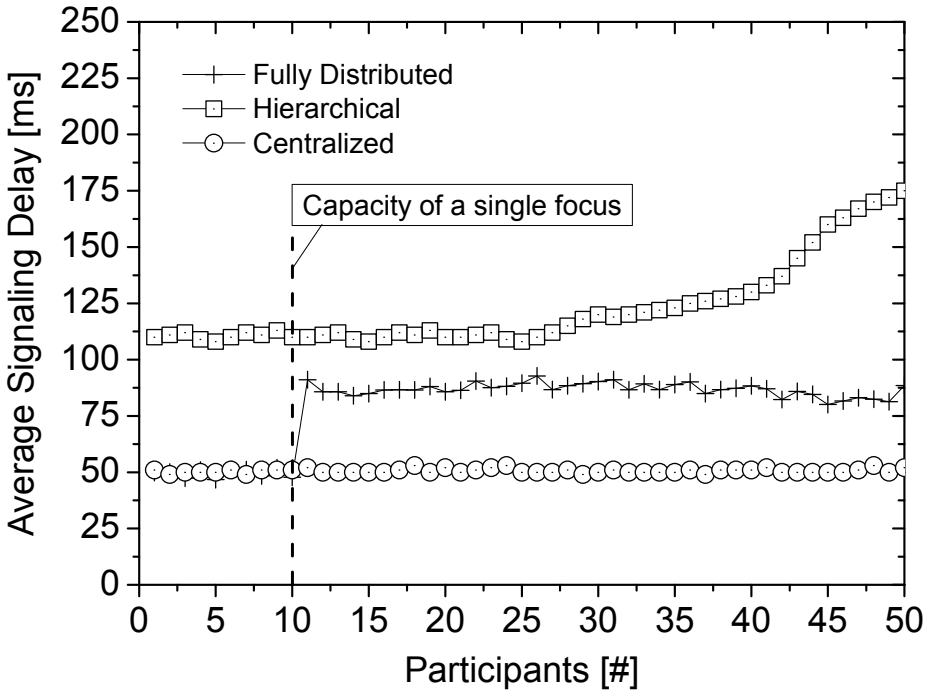

Figure 25: Signal Delay Average for SIP Invite Requests

6.1.3 Evaluation

These three measurement results are plotted in figure 25. As supposed, the test results for the centralized conferencing model keep a stable amount around $50ms$ per dialog establishment. The hierarchical approach by Cho et all begins at an amout about $110ms$, keeps it stable until reaching 30 participants, and afterwards slightly increases to an amount of $170ms$ signaling delay. The SDCON ratings, denoted as 'Fully Distributed', commence with an amount about $52ms$. The signal delays increase to an amount of $90ms$ for every new INVITE request after the 10th participation and remain constant in this band.

- Centralized Conferencing:

  The comparatively short measurement results for centralized conferencing originate from the direct connectivity to the focus node. A SIP INVITE request will be directly processed and responded. Because the amount for signaling do not require an high processing effort for focus entities, the time delay will remain stable. The issue for a centralized conference server arises by the costs for media mixing, that has to be provided simultaneous and the risk to be the single point of failure on server malfunction.

  Taken all values from the centralized measurement an signaling delay average with:

$$\bar{x} = \sum_{n=0}^{n} \frac{x_i}{n}$$

$$\bar{x} = 50,58ms$$

  will be calculated causing a variance with:

$$v = \frac{\sum_{n=0}^{n}(x_i - \bar{x})^2}{n-1}$$

$$v = 0,98$$

  with standard deviation with:

$$\sigma_x = \sqrt{Var(X)}$$

$$\sigma_x = 0,99$$

  These results illustrate the advantage of such a conferencing server model to serve a stable signaling to its clients with short delay times.

- Hierarchical Conferencing:

  Two different primary/regional focus topologies have been analyzed, a full mesh and a tree built, merging the results into one graph. The ratings of delay time when participants join a conference in the 'Hierarchical' are higher than in the centralized architecture. The authors do not explicitly comment about this, but it can be supposed that the higher effort of this architecture rely from the permission requests to the primary focus before a participants can join a conference. The continuously increasing delay is produced from following the cascade of regional focuses and mixer administration, as well as the participants reassignment to their regional focus in this distributed architecture [9].

  The signaling delay average for to join a conference in an hierarchical environment with

  $$\bar{x} = 123,04ms$$

  is more than double to the centralized architecture. This disadvantage will be compensated by a better scaling in media distribution to the conference members. Multiple focuses control multiple mixer server that will balance the media distribution effort per each mixing node. The average mixing count, a size to show how many media streams per participants are needed in at a single media distributor, increases linearly in a centralized server. In contradiction the load at a single mixer in the hierarchical system remains stable.

- Scalable Distributed Conferencing:

  This measurement have to be split into two operating modes during the conferencing process. First, the signaling delay while the conference where maintained by only one primary focus, and second, as multiple conference controllers are incorporated. As long as the number of members is ≤ 10, only one single focus is needed and operates as it would be a traditional centralized conference architecture. Hence, the measurement ratings are corresponding to those of the centralized conferencing architecture with an average:

  $$\bar{x} = 49,54ms$$

  with a variance of:

  $$v = 3,35$$

  and a standard deviation with:

  $$\sigma = 1,83$$

  The larger standard deviation in comparison to the centralized may result from fluctuating laboratory conditions where these ratings were made. The data packages from a participant are transmitted via a switch, that is frequently in use by other laboratories.

A second reason arises from the physical location of the focus peers to its corresponding clients. In most cases, the focus and its clients are located at the same device, but in some cases they are not because of the organic build of the conference topology. This causes one more network routing step between those focuses and their client.

For the measurements in the distributed operating mode, the following calculation results were obtained: A signaling delay average with:

$$\bar{x} = 90,25ms$$

with a variance of:

$$v = 8,33$$

that due to a standard deviation of:

$$\sigma = 2,89$$

These calculations results that the signaling delay is higher than in the centralized architecture. A lookup operation and one more routing step is needed to invite a new participant to the multiparty session.

In contrast to the hierarchical architecture the average to participate a conference remains constant. Independent from the underlying topology and conference size, a call delegation will be performed directly to the focus peers of free capacities. In the approach by Cho at all, a call delegation will be routed throughout all intermediate focus entities, e.g., in a tree topology $\lg N$ routing steps are needed, with N as the number of focus peers. The effort for assure the consistent and coherent conference status that offer the SDCON's lookup mechanism, does not affect the participations because they are achieved asynchronous. The high standard deviation can be caused by non-ideal laboratory conditions as explained above.

6.2 Dial-Out Operation

6.2.1 Measurement Setup

The second measurements were performed to analyze the average signaling delay for the dial-out operation. In detail, a time stamp was set whenever a conference member sends a SIP REFER to its focus peer, and stops the time as the INVITE request reaches the referred callee. This measurement was emulated on a single physical machine performing 20 independent repetitions. The schematic measurement setup is shown in figure 26. For this experiment 53 SDCON nodes have been instantiated, six as focus peers and 47 were running in standard SIP user agent mode. Like in the previous measurement, the focus capacity for serving clients have been adjusted to 10 participants with the demand to achieve a focus

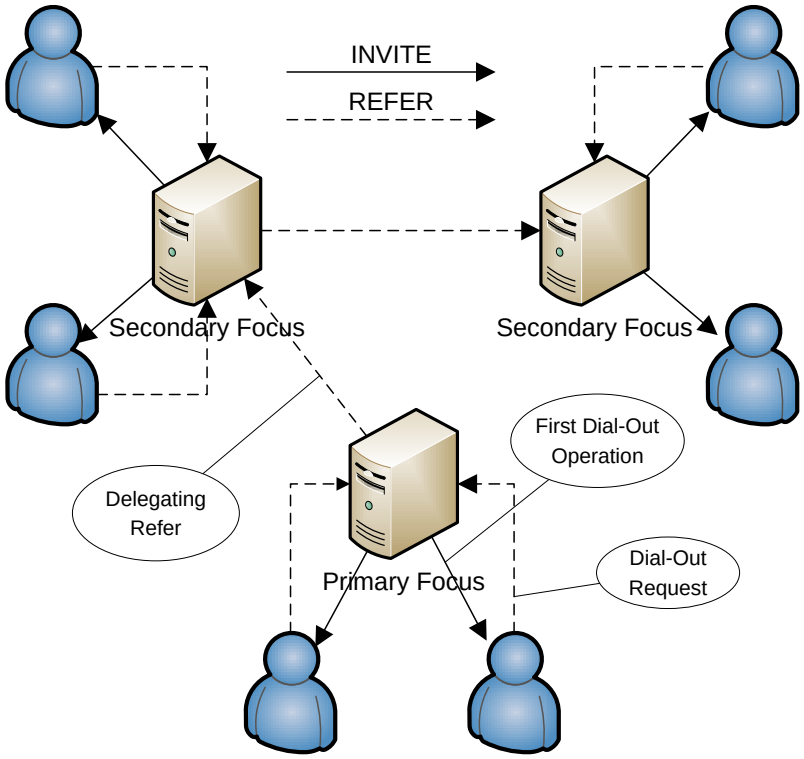

Figure 26: Referring new Participants

discovery on accepting the 9th call. To approximate network routing steps, a delay of $10ms$ per hop was added to the local measurements and like in the dial-in scenario, the measurement results where normalized to the centralized results for have a better comparison.

The behavior of the SDCON node were the following:

- 1. Peer: Startup and registering at its URI at a registrar
- 2. Peer: as 1. Peer and sending SIP INVITE to the 1. peer
- 3. Peer: Startup and register, also inviting the 1. peer. Because this invite causes the 1. peer to initiate the conference like described in section 2.2.2, the 3. Peer will be send a new INVITE towards the conference URI. Thereafter, it requests the focus peer to enforce the first dial-out operation the 4. peer sending SIP REFER.
- 4.-52. Peer: Startup and register. These peers are waiting for the focus's invitation to the conference and as achieved, they request the focus to invite the next participant.
- Last Peer: Just startup/register and waiting to be invited.

Participants #	Two-hop delay/ms
4	44,83
10	46,33
16	43,72
22	43,38
28	48,31
34	44,12
40	43,67
46	45,75
52	42,74

Table 2: Two hop dial-out Operation

Participants #	Three-hop delay/ms
3	56,6
11	62,26
20	62,6
29	61,12
38	60,19
47	60,98

Table 3: Three hop dial-out Operation

6.2.2 Measurement Results

Any focus peer that has not reached its `MAX_PARTICIPANTS` value, will accept the dial-out requests and directly invite the referee. Because this operation causes two routing steps it will be called 'Two-hop dial-out'. These results are listed in table 2. After the requested focus is fully booked, it will delegate the refers to a previews discovered focus with free capacity. Because this operation has three routing steps. it will be called 'Three-hop dial-out'. The measurements for this flows are shown in table 3.

In comparison to the previously demonstrated dial-in scenario, the advantage of distributed conference management becomes even more significant. A SIP REFER to a focus will be sent along the dialog route to the individual focus peers in contrast to an INVITE request to the conference URI, which always will firstly be routed through the primary focus. This behavior discharges the primary focus and for scenarios as described above. The majority of dial-out requests will be performed immediately as a Two-hop dial-out by a conference member's corresponding focus peer.

6.2.3 Evaluation

The measurement results are plotted in figure 27 and show the signaling delay for two-hop and three-hop dial-out. As expected, both scenarios exhibit a constant delay, whereas a two-hop dial-out needs about $45ms$ and at every 10th dial-out a three-hop action needs about $61ms$. An unexpected behavior is achieved by the 3rd participant's dial-out operation, which's duration is about $10ms$ longer than the other two-hop dial-outs. It is the first time that the focus receives a REFER request and because of this, it is expected that the underlying SIP stack has to create data structures at the implementation level that elongate the processing time. Another unexpected behavior can be noticed at the last two-hop result before a three-hop dial-out. The delays are enhanced by an amount of $2ms$ as compared to the other immediate

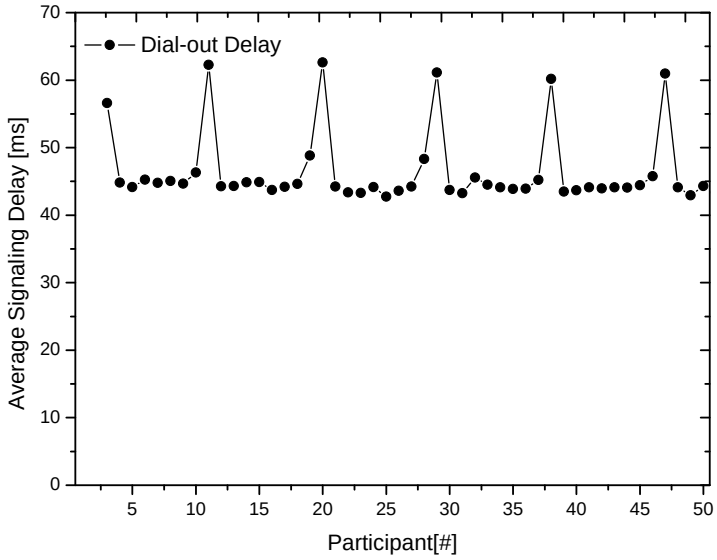

Figure 27: Signaling Delay for Dial-out Operation

refer operations. This effect again can be accounted to an additional data structure creation. At receiving the 9th REFER request, the focus has to memorize the corresponding refer dialog and corresponding server transaction for later discovery operations. This operation may cause this delay response behavior.

Taking these values and calculating them, the following results ensue for a two-hop dial-out:

A signaling delay average with:

$$\bar{x} = \sum_{n=0}^{n} \frac{x_i}{n}$$

$$\bar{x} = 44,4ms$$

will be calculated causing a variance with:

$$v = \frac{\sum_{n=0}^{n}(x_i - \bar{x})^2}{n-1}$$

$$v = 1,42$$

with standard deviation with:

$$\sigma_x = \sqrt{Var(X)}$$

$$\sigma_x = 1,19$$

These result display the constant character of a dial-out operation and sign the thesis of the previous measurement, which was executed on multiple physical devices, that the variabilities of the laboratories network and the different focus-to-client locations are causing a worse standard deviation.

Using the data values of the three-hop dial out, following results arise: A signaling delay average with:

$$\bar{x} = 63,43ms$$

with a variance of:

$$v = 0,97$$

that due to a standard deviation of:

$$\sigma = 0,99$$

The signaling delay average with about are $20ms$ longer than in the two-hop scenario originate from need of one more routing step and processing time at the remote focus peer. The result of the standard deviation emphasizes the stable character of such dial-out scenarios provided by the SDCON architecture.

In summary, dial-out operations seem to have an constant character and need shorter delay times, than dial-in requests to a distributed conference.

7 Conclusions and Outlook

This work presented an approach for large scale, distributed conference control in an ad hoc scenario using the Session Initiation Protocol in a peer-to-peer fashion. This approach relies on splitting the role of identifier and locator of the conference URI in a transparent and standard compliant fashion, and an extension for the event package for conference state at the application layer. The developed protocol schema has designed call flows for conference initiation, a way to discover potential controllers among conference members, and a call delegation mechanism that distributes conference management functions among the multiparty controllers. Such a distributed architecture addresses two challenges in conferencing. First, it establishes a base implementation for a distributed mixer topology, where the costs of media processing can be balanced among multiple entities. Second, it eliminates the single point of failure problem in traditional SIP conferencing by offering a coherent conference state information at each focus peer that can be used to compensate node failures.

These protocol modifications and conference operations were tested by using the standard-compliant SIP stack implementation JAIN SIP and provide a proof of concept for the call flows presented during this work as well as of the protocol conformal properties. The technique for conference control splitting was adopted by SIP user agents although they were unaware of this distributed environment.

The performance measurements displayed the operational aspects of the distributed conference management with about 50 participants by analyzing each in a dial-in and dial-out scenario.The signaling delays tend to remain in constant character in both scenarios.These results are in contrast to other distributed conferencing architectures that admit increasing delay times with a growing number of joining participants.

In future work the analysis of the protocol scheme should be extended and optimizations for this distributed system should be designed. The coherence of the conference state information at all controlling peers is a fundamental basis for performing distributed conferencing. Its carefully investigation and improvement will be a major task. An imported work to make this schema serviceable for a general consumer which's location often resides behind Internet-provider-adjusted router, a method for Network Address Translation (NAT)-traversal should be deployed. Therefore, it could be suggested to create a connectivity to the P2PSIP overlay network to perform protocols like STUN or ICE to traverse NATs and firewalls.

To display the full function of SDCON schema, an adoption to a peer-to-peer media streaming environment should be further designed. On may think of scenarios with multiple heterogeneous devices like smartphones, consumer computers and server systems participating one conference, but controlled by and served with media streams by various distinct endpoints. These roles in a conference could be adaptively chosen according to their capabilities and may seamlessly serve the multiparty approach.

Scalable, distributed conference control in tightly coupled SIP scenarios is still an exper-

imental work, but demonstrates the possibility to perform largely scaling conferences in an infrastructure-independent architecture.

A Appendix

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
        targetNamespace="http://www.example.org/inet-ci-multifocus-ext"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns="http://www.example.org/inet-ci-multifocus-ext"
        elementFormDefault="qualified" >
        <xs:element name="focus-states" type="focus-states-type"></xs:element>
        <xs:complexType name="focus-states-type">
                <xs:sequence>
                        <xs:element name="focus" type="focus-type"
                                maxOccurs="unbounded" minOccurs="0">
                        </xs:element>
                        <xs:any namespace="##other" processContents="lax"></xs:any>
                </xs:sequence>
                <xs:attribute name="state" type="state-type"></xs:attribute>
                <xs:anyAttribute namespace="##other" processContents="lax"></xs:anyAttribute>
        </xs:complexType>
        <xs:complexType name="focus-type">
                <xs:sequence>
                        <xs:element name="conf-id-holder" type="xs:boolean"
                                maxOccurs="1" minOccurs="0">
                        </xs:element>
                        <xs:element name="focus-capacity" type="focus-capacity-type"
                                maxOccurs="1" minOccurs="0">
                        </xs:element>
                        <xs:element name="participants" type="participants-type"
                                maxOccurs="unbounded" minOccurs="0">
                        </xs:element>
                        <xs:element name="graph" type="graph-type"
                                maxOccurs="unbounded" minOccurs="0">
                        </xs:element>
                        <xs:any namespace="##other" processContents="lax"></xs:any>
                </xs:sequence>
                <xs:attribute name="entity" type="xs:anyURI"></xs:attribute>
                <xs:attribute name="state" type="state-type"></xs:attribute>
                <xs:anyAttribute namespace="##other" processContents="lax"></xs:anyAttribute>
        </xs:complexType>
        <xs:complexType name="focus-capacity-type">
                <xs:sequence>
                        <xs:element name="max-participants" type="xs:int"
                                maxOccurs="1" minOccurs="0">
                        </xs:element>
                        <xs:element name="max-focus-references" type="xs:int"
                                maxOccurs="1" minOccurs="0">
                        </xs:element>
                        <xs:any namespace="##other" processContents="lax"></xs:any>
                </xs:sequence>
                <xs:attribute name="state" type="state-type"></xs:attribute>
                <xs:anyAttribute namespace="##other" processContents="lax"></xs:anyAttribute>
        </xs:complexType>
```

```
        <xs:complexType name="participants-type">
                <xs:sequence>
                        <xs:any namespace="##other" processContents="lax"></xs:any>
                </xs:sequence>
                <xs:attribute name="entity" type="xs:anyURI"></xs:attribute>
                <xs:attribute name="state" type="state-type"></xs:attribute>
                <xs:anyAttribute namespace="##other" processContents="lax"></xs:anyAttribute>
        </xs:complexType>
        <xs:complexType name="graph-type">
                <xs:sequence>
                        <xs:element name="ref-to-focus" type="xs:anyURI"
                                maxOccurs="1" minOccurs="0">
                        </xs:element>
                        <xs:any namespace="##other" processContents="lax"></xs:any>
                </xs:sequence>
                <xs:attribute name="state" type="state-type"></xs:attribute>
                <xs:anyAttribute namespace="##other" processContents="lax"></xs:anyAttribute>
        </xs:complexType>
</xs:schema>
```

Figure 28: Inet Multifocus Conference Information Extension XML Schema

References

[1] The Skype homepage. http://www.skype.com, 2009.

[2] F. Audet. The use of the SIPS URI Scheme in the Session Initiation Protocol (SIP). Internet Draft – work in progress 09, IETF, Nov. 2008.

[3] M. Barnes, C. Boulton, and O. Levin. A Framework for Centralized Conferencing. RFC 5239, IETF, June 2008.

[4] S. A. Baset and H. Schulzrinne. An analysis of the skype peer-to-peer internet telephony protocol. Technical report, IETF, sep 2004.

[5] D. Bryan. Concepts and Terminology for Peer to Peer SIP. Internet Draft – work in progress 00, IETF, July 2007.

[6] D. Bryan, P. Matthews, E. Shim, D. Willis, and S. Dawkins. Concepts and Terminology for Peer to Peer SIP. Internet Draft – work in progress 02, IETF, July 2008.

[7] G. Camarillo. The Internet Assigned Number Authority (IANA) Header Field Parameter Registry for the Session Initiation Protocol (SIP). RFC 3968, IETF, Dec. 2004.

[8] G. Camarillo, J. Ott, and K. Drage. The Binary Floor Control Protocol (BFCP). RFC 4582, IETF, Nov. 2006.

[9] Y.-H. Cho, M.-S. Jeong, J.-W. Nah, W.-H. Lee, and J.-T. Park. Policy-Based Distributed Management Architecture for Large-Scale Enterprise Conferencing Service Using SIP. *Selected Areas in Communications, IEEE Journal on*, 23(10):1934–1949, Oct. 2005.

[10] D. Cohen, S. Casner, and J. W. Forgie. A network voice protocol nvp-ii. Technical report, U S C / I S I, apr 1981.

[11] H. L. Cycon, T. C. Schmidt, G. Hege, M. Wählisch, D. Marpe, and M. Palkow. Peer-to-Peer Videoconferencing with H.264 Software Codec for Mobiles. In R. Jain and M. Kumar, editors, *WoWMoM08 – The 9th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks – Workshop on Mobile Video Delivery (MoViD)*, pages 1–6, Piscataway, NJ, USA, June 2008. IEEE, IEEE Press.

[12] L. Delgrossi and L. Berger. Internet Stream Protocol Version 2 (ST2) Protocol Specification - Version ST2+. RFC 1819, IETF, Aug. 1995.

[13] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, IETF, Aug. 2008.

[14] I. Dorros. Picturephone. *Bell Laboratories Records*, 47, 1969.

[15] N. Freed and N. Borenstein. Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples. RFC 2049, IETF, Nov. 1996.

[16] M. Handley and V. Jacobson. SDP: Session Description Protocol. RFC 2327, IETF, Apr. 1998. Obsoleted by RFC 4566, updated by RFC 3266.

[17] ITU-T Recommendation H.264 & ISO/IEC 14496-10 AVC. Advanced Video Coding for Generic Audiovisual Services. Technical report, ITU, 2005. Draft Version 3.

[18] ITU-T Recommendation H.323. Infrastructure of audio-visual services - Systems and terminal equipment for audio-visual services: Packet-based multimedia communications systems. Technical report, ITU, 2000. Draft Version 4.

[19] The NIST JAIN-SIP homepage. http://jain-sip.dev.java.net/, 2009.

[20] The JAXB Workshop homepage. https://jaxb-workshop.dev.java.net/, 2009.

[21] C. Jennings, B. Lowekamp, E. Rescorla, S. Baset, and H. Schulzrinne. REsource LOcation And Discovery (RELOAD) Base Protocol. Internet Draft – work in progress 01, IETF, Dec. 2008.

[22] A. Johnston and O. Levin. Session Initiation Protocol (SIP) Call Control - Conferencing for User Agents. RFC 4579, IETF, Aug. 2006.

[23] A. Johnston, R. Sparks, C. Cunningham, S. Donovan, and K. Summers. Session Initiation Protocol Service Examples. RFC 5359, IETF, Oct. 2008.

[24] A. Knauf, T. C. Schmidt, and M. Wählisch. Scalable, Distributed Conference Control in Heterogeneous Peer-to-Peer Scenarios with SIP. In M. Younis and C. T. Chou, editors, *Proc. of the 5th International Mobile Multimedia Communications Conference (MobiMedia)*, Lecture Notes of ICST, Berlin, Heidelberg, September 2009. Springer Verlag. Accepted for publication.

[25] O. Levin and R. Even. High-Level Requirements for Tightly Coupled SIP Conferencing. RFC 4245, IETF, Nov. 2005.

[26] R. Mahy and D. Petrie. The Session Initiation Protocol (SIP) "Join" Header. RFC 3911, IETF, Oct. 2004.

[27] P. O'Doherty and M. Ranganathan. Jain sip tutorial. serving the developer community. http://snad.ncsl.nist.gov/proj/iptel/tutorial/JAIN-SIP-Tutorialv2.pdf, 2003.

[28] J. Ott, C. Perkins, and D. Kutscher. Requirements for Local Conference Control. Internet Draft – work in progress 00, IETF, Dec. 2000.

[29] M. Palkow. The daViKo homepage, 2009. http://www.daviko.com.

[30] C. B. Peters. Talks on 'see-phone'. *The New York Times*, September 18 1938.

[31] A. B. Roach. Session Initiation Protocol (SIP)-Specific Event Notification. RFC 3265, IETF, June 2002. Updated by RFC 5367.

[32] S. Romano, A. Amirante, T. Castaldi, L. Miniero, and A. Buono. A Framework for Distributed Conferencing. Internet Draft – work in progress 04, IETF, Dec. 2008.

[33] S. Romano, A. Amirante, T. Castaldi, L. Miniero, and A. Buono. Requirements for Distributed Conferencing. Internet Draft – work in progress 04, IETF, Dec. 2008.

[34] S. Romano, A. Amirante, T. Castaldi, L. Miniero, and A. Buono. Requirements for the XCON-DCON Synchronization Protocol. Internet Draft – work in progress 04, IETF, Dec. 2008.

[35] J. Rosenberg. A Framework for Conferencing with the Session Initiation Protocol (SIP). RFC 4353, IETF, Feb. 2006.

[36] J. Rosenberg. Obtaining and Using Globally Routable User Agent (UA) URIs (GRUU) in the Session Initiation Protocol (SIP). Internet Draft – work in progress 15, IETF, Oct. 2007.

[37] J. Rosenberg, J. Peterson, H. Schulzrinne, and G. Camarillo. Best Current Practices for Third Party Call Control (3pcc) in the Session Initiation Protocol (SIP). RFC 3725, IETF, Apr. 2004.

[38] J. Rosenberg and H. Schulzrinne. An Offer/Answer Model with Session Description Protocol (SDP). RFC 3264, IETF, June 2002.

[39] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261, IETF, June 2002. Updated by RFCs 3265, 3853, 4320, 4916, 5393.

[40] J. Rosenberg, H. Schulzrinne, and O. Levin. A Session Initiation Protocol (SIP) Event Package for Conference State. RFC 4575, IETF, Aug. 2006.

[41] R. Sparks. The Session Initiation Protocol (SIP) Refer Method. RFC 3515, IETF, Apr. 2003.

[42] A. S. Tanenbaum and M. van Steen. *Distributed Systems: Principles and Paradigms (2nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006.

[43] S. S. Team. Skype support request: Maximum conference sizes in skype overlay. https://support.skype.com/de/, July 2009.

[44] C. Topolcic. Experimental Internet Stream Protocol: Version 2 (ST-II). RFC 1190, IETF, Oct. 1990. Obsoleted by RFC 1819.

[45] Xml schema part 1: Structures second edition. W3C Recommendation, World Wide Web Consortium, October 2004.

[46] Xml schema part 2: Datatypes second edition. W3C Recommendation, World Wide Web Consortium, October 2004.

[47] M. Zangrilli and D. Bryan. A Chord-based DHT for Resource Lookup in P2PSIP. Internet Draft – work in progress 00, IETF, Feb. 2007.

List of Figures

Abbreviations

ACK	SIP Acknowledgment
AOR	Address-of-Record
BFCD	Binary Floor Control Protocol
DCON	Distributed Conferencing
DHT	Distributed Hash Table
DNS	Domain Name System
GRUU	Globally Routable User agent URI
GUI	Graphical User Interface
IANA	Internet Assigned Number Authority
ICE	Interactive Connectivity Establishment
IDL	Interface Definition Language
IETF	Internet Engineering Task Force
IFA	Internationale Funkaustellung in Berlin
IM	Instant Messenger
IP	Internet Protocol
ISDN	Integrated Services Digital Network
JAIN	Java APIs for Integrated Network
JAXB	Java Architecture for XML Binding
MCU	Multipoint Control Unit
MIME	Multipurpose Internet Mail Extensions
NAT	Network Address Translation
NVP-II	Network Voice Protocol
P2P	Peer-to-Peer
P2PSIP	Peer-to-Peer SIP
PVP	Packet Video Protocol
RELOAD	REsource LOcation And Discovery
SDCON	Scalable Distributed Conferencing
SDP	Session Description Protocol
SIP	Session Initiation Protocol
SIPPING	Session Initiation Proposal Investigation WG
ST/ST2	Internet Stream Protocol
STUN	Session Traversal Utilities for NAT
UA	User Agent
URI	Uniform Resource Identifier
VoIP	Voice over IP
VVoIP	Voice and Video over IP
WG	IETF Working Group
XCON	Centralized Conferencing

XDSP ..... XCON-DCON Synchronization Protocol
XJC ..... XML Java Compiler
XML ..... Extensible Markup Language

Listings

List of Tables

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) bzw.§24(4) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, August 25, 2009 Alexander Knauf