A Conference Control Model for Light-weight Sessions

(DRAFT)

Woohyong Choi

A thesis submitted to the faculty of the Korea Advanced Institute of Science and Technology in partial fulfillment of the requirements for the degree of Master of Science in the Department of Computer Science.

July 1, 1996

Korea Advanced Institute of Science and Technology

ABSTRACT

The current model for light-weight multicast session based teleconferencing applications provide a very primitive set of control mechanisms such as *net mutes mic* and *mic mutes net*. Commercial products based on T.124 Recommendation of International Telecommunication Union(ITU) are being introduced, and it seems likely that a similar one based on T.124 will be derived for Internet usage. However, the current emphasis on wide area scalable multicast based conferencing in the Internet Engineering Task Force(IETF) is desirable, and we shouldn't sacrifice the benefits of multicast based sessions to conform to the tightly coupled model of T.124 Recommendation.

This paper proposes a conference control model for light-weight sessions where media applications can collaborate with a coordination tool to provide a level of control over the light-weight sessions. This coordination tool provides a generic base to manage conferencing states, find agreements among the participants upon which a varying range of policies could be implemented without any changes to each applications. A prototype of the coordination tool has been built and is being used to provide conference control to existing applications.

CONTENTS

LIST OF FIGURES

Chapter 1

INTRODUCTION

There has been much work in recent years on multimedia teleconferencing applications based on desktop computers. Significant progress has been made on enabling technologies such as packet transport for audio and video, resource management, connection establishment, scalability and privacy. These applications are now highly usable and, after long gestating as research prototypes, are now vigorously entering the mainstream commercial market. With the increasing availability of audio and video equipment on workstations, and with much faster networks being installed, we expect teleconferencing applications to become an important component of many future social and business interactions.

The previous generation of conferencing tools, such as mmconf[2], etherphone[22] and the Touring Machine[3] were based on centralized architectures, where a central application on a central machine acted as the repository for all information relating to the conference. Although simple to understand and simple to implement, this model proved to have a number of disadvantages, the most important of which was the disregard for the failure modes arising from conferencing over the wide area[10].

Since early 1992, a multicast virtual network has been constructed over the Internet[1]. This multicast backbone or MBone[14] has been used for

a number of applications including multimedia (audio, video and shared workspace) conferencing. The initial deployment of these applications such as vat[12], ivs[20], nv[4], and vic[15] has proven successful especially in terms of scalability. This alternative approach to the centralized model is the lightweight session model promoted by Van Jacobson[13]. In the lightweight session model, communication is regarded as inherently unreliable and applications are loosely-coupled cooperating instantiations distributed over the network. Observations of the MBone show that humans can cope with a degree of inconsistency that arises from partitioned networks and lost messages, as long as the distributed state will tend to converge in time. However, they just lend themselves to relatively open conferences, and do not have ways to support the various range of policies in diverse social conventions[19].

Recently, centralized conference systems based on International Telecommuniaction Union(ITU)'s T.124[21] recommendation are being introduced into market places, and it seems likely that a similar one based on T.124 will be derived for internet usage[7]. However, the current emphasis of the Internet Engineering Task Force(IETF) on wide area scalable multicast based conferencing is desirable, and we shouldn't sacrifice the benefits of multicast based sessions to conform to the ITU centralized model[6].

This paper presents a conference control model for the light-weight sessions. The model relies on a coordinator running on each conferees' host to manage shared states among the participants and to control media applications. Policies are specified in a way that they are not to tied to any specific mechanisms so that policies can be easily changed as needed. To manage shared states, the coordinator use an agreement protocol derived from a similar protocol[19] proposed by the Multiparty Multimedia Session Control(MMUSIC) working group[17] of Internet Engineering Task Force.

Coordination among media applications and the coordination tool are made possible by the conference bus abstraction as briefly introduced in the design of vic[15]. An implementation of the conference tool has been demonstrated in several scenarios, in particular, negotiation of media encoding, and floor management.

Chapter 2

RELATED WORKS

In the following sections, some of the related works in the area of conference control model are discussed to get the requirements for the conference control model. MBone applications are discussed as the paper is about to add conference control to these applications. It is meaningful to discuss Conference Control Channel Protocol[10] since it proposes a conferencing architecture with requirements from MBone applications as well as from the MICE[9] project.

The task of conference control breaks down in the following way[10].

Application Control Applications need to be started with the correct initial state, and the knowledge of their existance must be propagated across all participating sites. Applications may need to cooperate(for example to archive audio and video synchronization).

Membership Control Who is currently in the conference and has access to what applications.

Floor Management Who or what has control over the input to particular applications.

Network Management Requests to set up and tear down media connections between end-points (no matter whether they be analogue through a

video switch, a request to set up an ATM virtual circuit, or using RSVP[23] over the Internet), and requests from the network to change bandwidth usage because of congestion.

Meta-conference Management How to initiate and finish conferences, how to advertise their availability, and how to invite people to join.

It is observed that the problem of meta-conference management is outside the bounds of the conference control architecture, and should be addressed using tools such as Lawrence Berkeley Laboratory(LBL)'s Session Directory[sd], traditional directory services or through external mechanisms such as email. The conference control system is intended to maintain consistency of state amongst the participants as far as is practical and not to address the social issues of how to bring people together, and co-ordinate initial information.

It is also believed that the membership control is the problem of limiting and/or modifying participation and is entirely a key distribution/revocation problem[13]. This same problem appears in many other areas of the internet architecture, not just conferencing (network management, resource reservation, online publication, etc.).

This paper doesn't attempt to define an exhaustive set of session control protocol to support all the tasks described above. Instead, the paper take these tasks as the basis for defining a conference control model upon which a set of simple messages are exchanged between conferencing applications.. The model primarily focus on providing floor management and application control. Conference control policies are configurable and policies are separated from mechanisms so that the policies can be changed as required.

2.1 MBone Applications

Since early 1992, a multicast virtual network has been constructed over the Internet[1]. This multicast backbone or MBone [14] has been used for a number of applications including multimedia (audio, video and shared workspace) conferencing. These applications involved include vat[12], ivs[20], nv[4], and wb[11] amongst others. These applications have a number of things in common:

- They are based on IP multicast.
- They all report who is present in a conference by occasional multicasting of session information.
- The different media are represented by separate applications.
- There is no conference control, other than each site deciding when and at what rate they send.

The light-weight session model used in these application as promoted by Van Jacobson[13] is an alternative approach to the traditional centralized model. Communication is regarded as inherently unreliable and applications are loosely-coupled cooperating instantiations distributed over the network.

These applications are designed so that conferencing will scale effectively to large members of conferees. At the time of this writing, they have been used to provide audio, video and shared whiteboard to conference with about 500 participants. Without multicast, this is clearly not possible. It is also clear that, with unreliable networks, these applications cannot achieve complete consistency between all participants, and so they do not attempt to do so – the conference control they support usually consists of:

- Periodic (unreliable) multicast reports of receivers
- The ability to locally mute a sender if you do not wish to hear or see them. However, in some cases stopping the transmission at the sender is actually what is required.

Thus any form of conference control that is to work with these applications should at least provide these basic facilities, and should also have scaling properties that are no worse than the media applications themselves.

The domains these applications have been applied to vary immensely. The same tools are used for small (say 20 participants), highly interactive conferences as for large (500 participants) disseminations of seminars, and the applications developers are working towards being able to use these applications for *broadcasts* that scale towards millions of receivers.

It should be clear that any proposed conference control scheme should not restrict the applicability of the applications it controls, and therefore should not impose any single conference control policy. For example we would like to be able to use the same audio encoding engine (such as vat), irrespective of the size of the conference or the conference control scheme imposed.

Since the various media in a conference session are handled by separate applications, we need a mechanism to provide coordination among the separate processes. The *Conference Bus*[15] abstraction, illustrated in figure 2.1, provides this mechanism. The concept is simple. Each application can broadcast a typed message on the bus and all applications that are registered to receive that message type will get a copy. The figure depicts a single session composed of audio(`vat`), video(`vic`), and whiteboard(`wb`) media, orchestrated by a (to be discussed in this paper) coordination tool(`ct`).

The conference bus is currently only used to support voice-switched windows in vic with cues from vat to focus on the current speaker. The bus

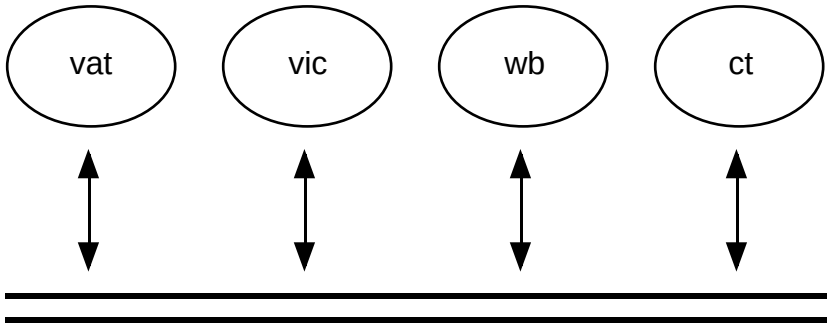

Fig. 2.1: The Conference Bus

can be utilized to support floor control, synchronization among the various media applications, and to support coordinated access to audio and video devices.

Conference buses are implemented as multicast datagram sockets bound to the loopback interface. Local-machine IP multicast provides a simple, efficient way for one process to send information to arbitrary set of processes without needing to have the destinations *wired in*. Since one user may be participating in several conferences simultaneously, the transport address (UDP destination port) is used to create a separate bus for each active conference. This simplifies the communication model since a tool knows that everything it sends and receives over the bus refers to the conference it is participating in and also improves performance since tools are awakened only when there is activity in their conference. Each application in the conference is handled the address (port) of its bus via a startup command line argument.

2.2 Conference Control Channel Protocol

The Conference Control Channel Protocol[10] abstracts a messaging channel providing reliable/unreliable semantics using a simple distributed interprocess communication system. The protocol defines a simple class hierarchy,

with an application type as the parent class and subclasses of network manager, member and floor manager, and define a generic protocols that are used to talk between these classes and the application class, and an inter-application announcement protocol.

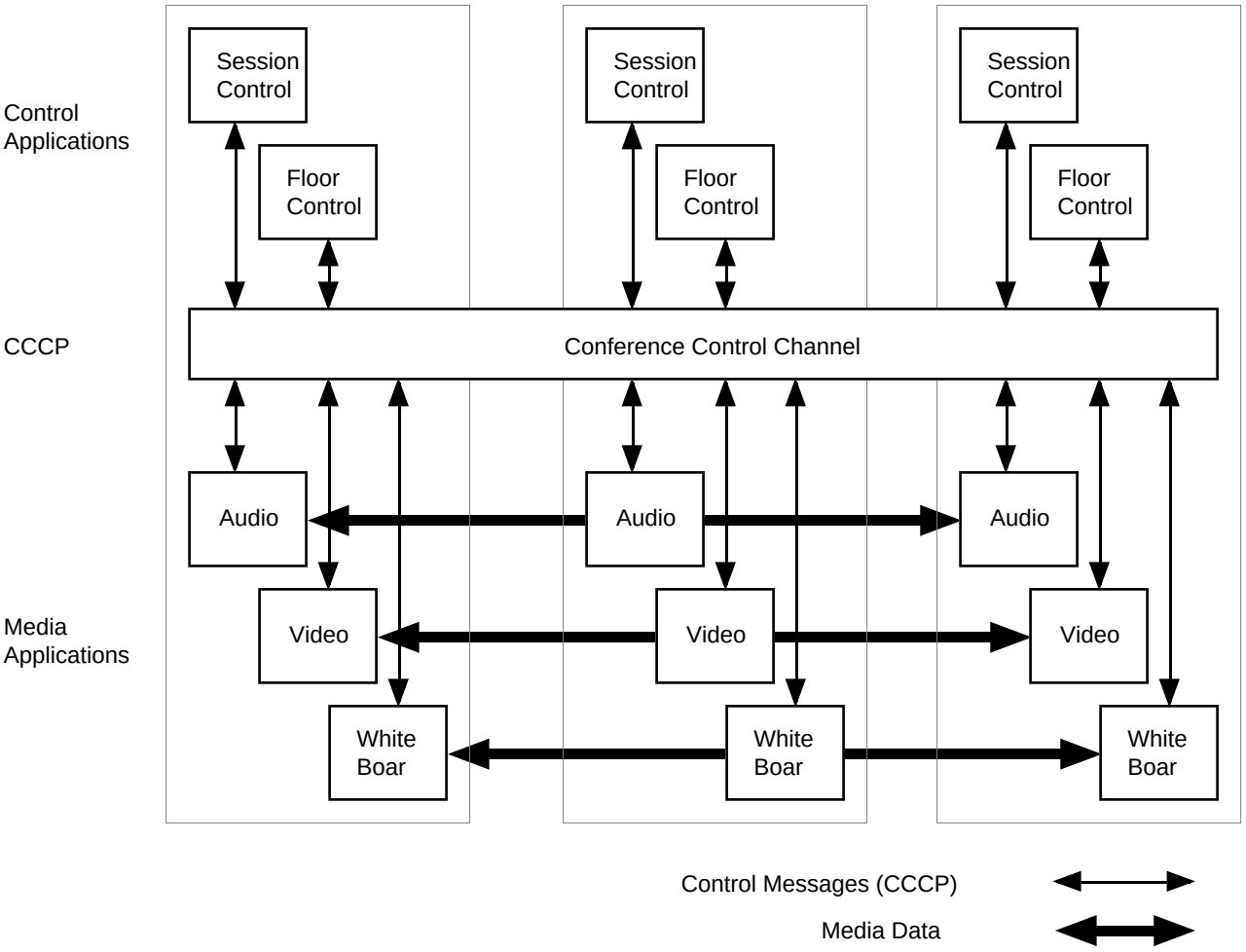

Fig. 2.2: Conceptualisation of CCCP

CCCP takes the following requirements originated primarily from the MICE[9] project as well as from multicast internet conferencing.

Modularity Conference Control Mechanisms and Conference Control applications should be separated. The mechanism to control applications

(mute, unmute, change video quality, start sending, etc) should not be tied to any one conference control application. This suggests that a modular approach be taken, with for example, specific floor control modules being added when required.

A unified user interface Each applications in multicast internet conferencing has a separate set of session information. For example, a participant in a conference using vat(audio), vic(video), and wb(whiteboard) has three separate sets of session information, and three places to look to see who is active. CCCP states that any conference interface should provide a single set of session and activity information. Say, a conferee wants to mute the other user in a conference, this should be possible from a single interface with a single command.

Flexible floor control policies Conferences come in all shapes and sizes. It should be possible to provide floor control functionality, but the providers of audio, video and workspace applications should not specify which policy to be used.

Scaling from tightly coupled to loosely coupled conferences CCCP originates in part as a result of experience gained from tightly coupled centralized systems, such as the Touring Machine system[3] and also from MBone based loosely coupled conferences. Tightly coupled conferences have advantages for small conferences where membership needs to be controlled. Loosely coupled conferences are the only way to achieve scalability, but the current lightweight sessions are too unrestricted for some uses.

Chapter 3

MODEL

We now address the conference control model that will support requirements discusses in the previous chapter. Observations from the previous chapter lead us to the conclusion that the conference control model for light-weight sessions has the following requirements.

- Existing applications be used without much modifications
- Retain scaling properties of media applications
- Support flexible policies

The conference control model discussed in the paper should work with existing applications in the Internet multicast conferencing. Thus any form of conference control that is to work with these applications should at least provide these basic facilities, and should also have scaling properties that are no worse than the media applications themselves. It should also be clear that any proposed conference control scheme should not restrict the applicability of the applications it controls, and therefore should not impose any single conference control policy. A class of ten year olds might use very different floor control from a class of PhD candidates.

The conference control model discussed in this paper take the MMUSIC agreement protocol[19] as an integral part of the architecture. Before we go

into the further details of the model, the agreement protocol needs to be discussed briefly.

3.1 Agreement Protocol

The agreement protocol introduces a framework for expressing a broad family of policies for joint control of ephemeral state. These policies describe who can propose changes to state, and the degree of consensus needed to enact these changes. The policies also describe to what extent the views of state must be consistent when voting and when all changes to state have been executed.

The protocol consider two different models of the reliability of communication: reliable and unreliable. In the reliable communication model, all messages arrive at their intended destination on time and all messages from a particular sender to a particular receiver arrive in the order in which they are sent; we do not require that messages from different senders arrive in the same order as sent. In the unreliable model, the probability that k consecutive messages from a sender destined for a particular receiver all fail to arrive is $O(k^{-y})$. In this model, messages from the same sender may arrive out of order.

The protocol also consider two different models for how the underlying transport is addressed. In the Explicitly Named List (ENL) model, messages are delivered only to those members that are known by the sender to be members. Another would be using a sender-defined multicast tree. In the Shared Bus (SB) model, messages are delivered to all those who choose to receive the messages. each model has two different degree of reliability, reliable and unreliable. Combining the reliability and addressing options gives four systems to consider; reliable ENL, unreliable ENL, reliable SB, and un-

reliable SB. The internet multicast where the light-weight session model is based on can be described as "unreliable SB" model. We now restrict our discussions with the unreliable SB model in mind.

Policies are specified along three dimensions: initiator policies, voting policies, and consistency policies.

initiator For each elemental operation O_i we define the initiating set of members $I(O_i)$; only those members in $I(O_i)$ can initiate the operation.[1]

voting For every elemental change operation O_i, there is an associated voting rule $V(O_i)$ which takes a vector of votes, with each element taking a value from {YES, NO, ABSTAIN, NO-REPLY} and returns 0 if the vote fails and returns 1 if the vote carries. Voting rules can span the gamut from requiring unanimous consent to requiring no consent at all. The first three responses (YES, NO, ABSTAIN) are explicit responses from other members; the NO-REPLY value indicates that no explicit response was received.

consistency This dimension is of particular importance when, as assumed in this paper, the teleconference is to be controlled in a distributed manner. There is no other definition of the session state besides the state belonging to each member; in particular, there is no "truth" against which to compare these individual versions. We can only compare these individual versions to each other and require various degree of consistency. The weak eventual consistency requires only that the state on

[1] The initiator policy described in the MMUSIC agreement protocol required that changes on variables can only be initiated by a certain specified member. This should be relaxed to *changes on variables can only be initiated by a certain specified member at a given moment.* The relaxed requirement allows initiators to be changed over time.

which the eventual members agree be given by a sequence of change operations that may be only a subset of all executed change operations; that is, some executed change operations may be ignored, as long as all members ignore them[2].

There are three dimensions to policy: initiation, voting and consistency. The implementation of the initiation rules is essentially a local matter. We use repeated transmission of a State message that announces the current value of a particular (or a set of) state variables. We choose to announce the resulting variable, rather than the operation, since (as will become clearer below) that relieves the need for all members to receive exactly the same set of change operations to eventually agree. Our mechanisms in the unreliable cases rely on the set of messages:

Poll(Id: id, Operation: op, Variable: i, Value: value-i, Variable: j, Value: value-j, ...)
Asks for a vote on the proposed operation, with result as shown. The response is {YES, NO, ABSTAIN}.

Response(Id: id, Response: response) Response to Poll message.

State(Variable: i, Value: value-i, Variable: j, Value: value-j, ..., Time: timestamp)
This announces the content of state variable i (several state variables can be included in the same message).

[2] Consistency policies include consistent voting and strong eventual consistency. Since they cannot be implemented in the unreliable communication model, we do not consider these policies in the discussion. We do not attempt to build a reliable transport protocol on top of the unreliable communication media, which will take discussions back to reliable communication. The design of the reliable communication protocol is out of the scope of this paper

If the proposed operation requires a vote, then the message exchange is Poll, Response, and then a sequence of State messages. If no vote is required, then there is merely the sequence of State messages. Exactly how this sequence of State messages is sent out will determine the overhead of the algorithm and its correctness.

3.2 Conference Control Architecture

In theory, the basic conference control architecture is straightforward: we establish a conference bus[15] to exchange messages between media applications and coordination tool. Coordination tool dictates behaviors of media applications as defined in the session policy; coordination tool can send mute message or unmute message to media applications. Session policies are described in series of procedural commands that are interpreted by coordination tool. To take some examples, Poll command to cast a vote among the conferees and Send command to send messages to the media applications via the conference bus.

In the following subsections, we look into more details each parts that constitutes the conference control architecture

3.2.1 Coordination Tool

The coordination tool lies in the central part of the conference control model. The coordination tool keeps consistencies of shared variables and make changes to variables as defined by the policy description. Upon changes to certain variables, the coordination tool puts messages on the conference bus so that media applications listening to the conference bus to take appropriate actions according to the messages.

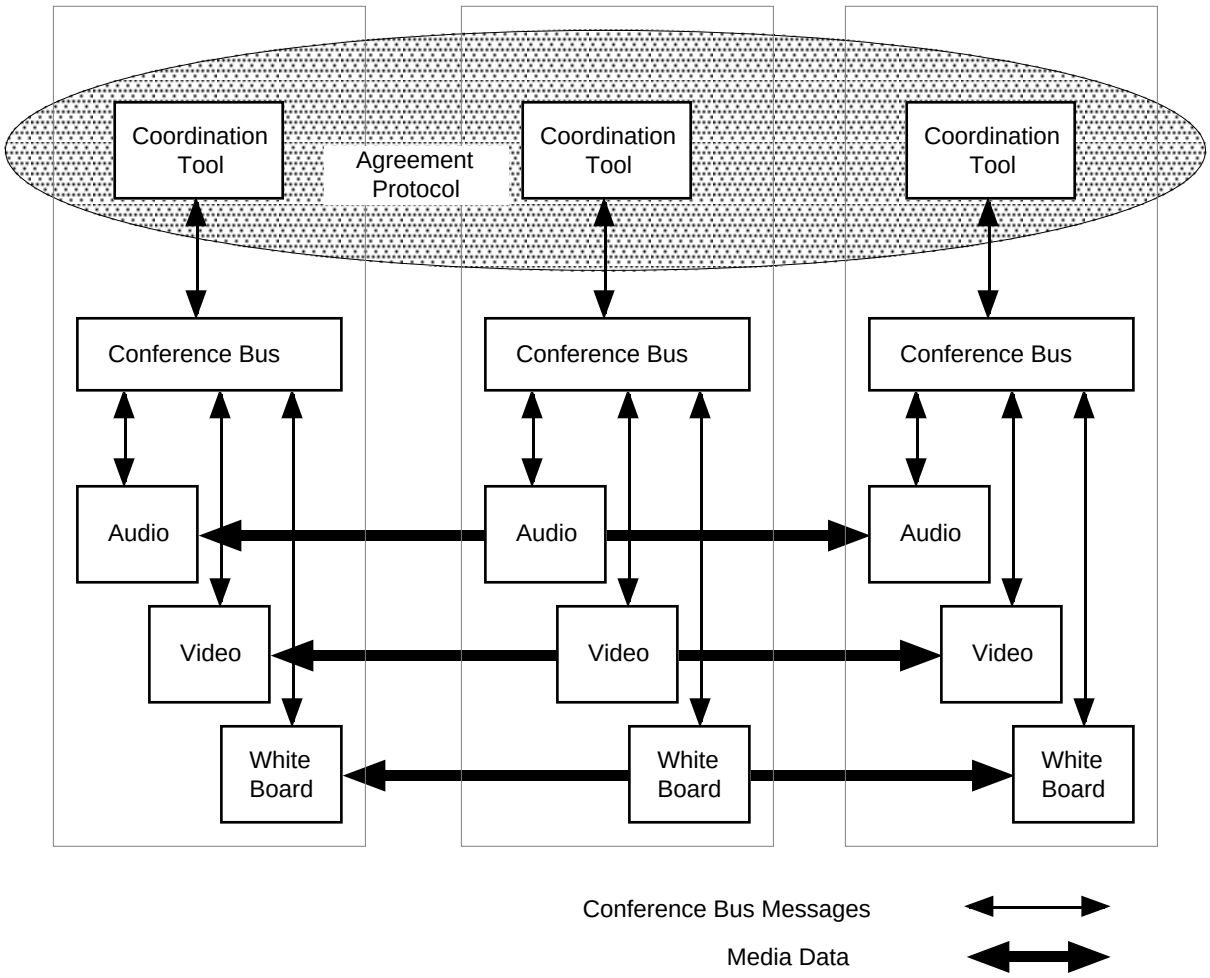

Fig. 3.1: The Conference Contrl Model

There are two kinds of communication in the coordination tool; communication with media applications is implemented through conference buses and communication with other coordination tools are implemented as multicast sockets bound to another transport address of the current conference. The agreement protocol is bound on the latter part of the communication interface.

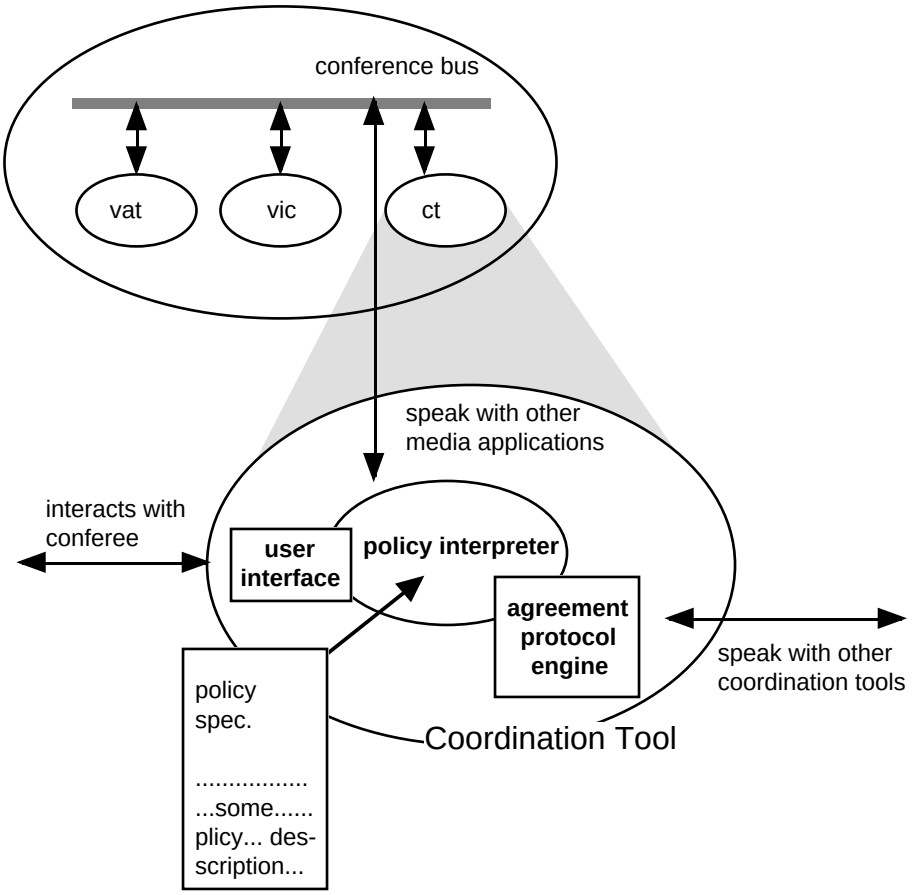

Fig. 3.2: The Coordination Tool

3.2.2 Conference Bus

An application listens to the conference bus and find if any message is addressed to itself by comparing the message with the list of know messages defined within the application. The messages are predefined, and applications need to be configured to handle the messages.

The coordination tool can send messages to media applications through the conference bus to assure that the media application follow any specific session policy of the conference; the coordination tool can send mute message or unmute message to media applications. Media applications also can send

messages to the coordination tool; vat(audio) asking for *floor* to the coordination tool. Media applications themselves can send messages to each other to achieve cross-media synchronization. Messages in the conference bus can be defined as needed.

Media applications as well as the coordination tool should be designed so that any message handle could be easily incorporated them. Vat, vic, and wb currently support this by separating user interface part with the core application design.

3.2.3 Specifying Policies

There are three dimensions to policy: initiation, voting and consistency. Any change operation that doesn't require a vote can be modeled as a voting with *pass always* condition and any initiation policy can be combined into a vote cast. The consistency is always supported by the agreement protocol. This lead us to the fact that any policy statement can be specified as a form of a vote. A policy statement can now be specified as a tuple of the following variables.

> Poll(label, initiator, pass-condition, query-dialogue, notify-dialogue, pass-code, reject-dialogue, fail-code, variable i, value value-i, variable j, value value-j, ...)

label Identifier string to be mapped with the user interface object.

initiator who can initiate this poll

pass-condition the condition that should be met for the poll to be passed

query-dialogue query dialogue to be used in interacting with a conferee

notify-dialogue notification dialogue to be shown to all conferees when the poll is passed

reject-dialogue notification dialogue to be shown to the initiator when the poll cannot be passed.

pass-code script to be executed when the poll passes

fail-code script to be executed when the poll fails

Chapter 4

DESIGN

This chapter discusses a coordination control tool prototyped from the model described in the previous chapter. Before we go into the details of the coordination tool, programming languages issues are first discussed.

The policy specification to be used in coordination tool should support the following features.

Graphical user interface Conferencing is fundamentally a human endeavor and the policy specification involves interactions with users.

Messaging To send messages over the conference bus and to the other coordination tools in the conference.

Standard procedural language features It seems hard to specify policies other than procedurally[13].

To meet these requirements, the policy specification is implemented as a Tcl/Tk script[16]. Tcl is a simple scripting language which was originally developed as a generic command language for intergated circuit design. Tcl provides generic programming facilities, such as variables and loops and procedures, that are useful for a variety of applications. Furthermore, its interpreter is a library of C procedures that can easily be incorporated into

applications, and each application can extend the core Tcl features with additional commands for that application. One of the most useful extensions to Tcl is Tk, which is a toolkit for the X window System. Together with Tcl, Tk provide a programming system for developing and using graphical user interface. We use a distributed programming extension to Tcl/Tk, called Tcl-DP for the development of the coordination tool.

4.1 Coordination Tool

The coordination tool is composed of the following parts.

Agreement Protocol Engine Keeps the states in the coordination tool consistent with the states in the other coordination tools. In the events of changes in variables, appropriate messages are sent to the conference bus to control media applications.

Policy Interpreter Interpretes the policy specified in Tcl commands and hooks up policies with user interface glues.

User Interface Displays participants of the conference, states of the variables, and debug messages. Prompts dialog boxes when the coordination tool need users' attention.

When the coordination tool is initiated, the policy interpreter parses a policy specification, and binds each policies with a user interface object.

4.1.1 Runtime Description

User casts a vote by selecting a menu item under the *Policy* menu bar. The command bound with the menu item is executed and the initiator sends

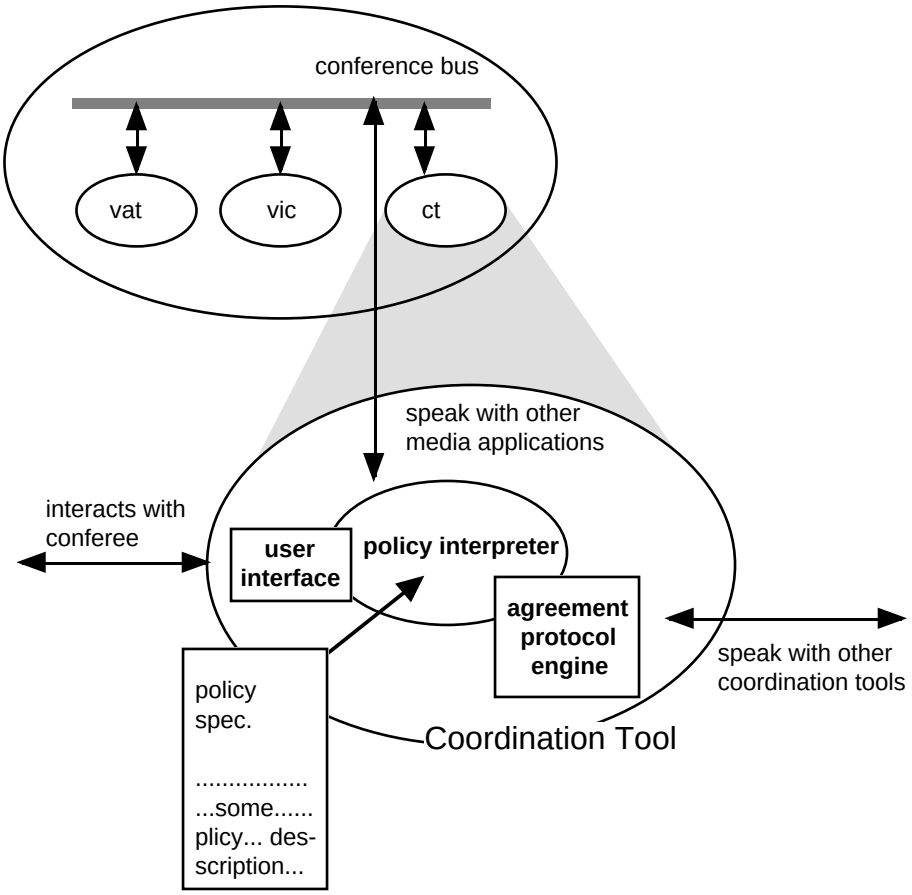

Fig. 4.1: The Coordination Tool

poll message to relevant conferees. Conferees participating in the vote are prompted with a dialog asking whether they agree the vote to be passed or not. Upon receiving user response, each conferee's coordination tool responds with response messages. The initiator solicits response messages for a certain amount of time[1] and checks if the poll condition is passed each time it receives a response message. If the poll passes, the initiator sends out state message periodically. Figure 4.5 depicts the dynamics of the coordination tool.

[1] the time is a linear function of *time-to-live* value of a conferencing session

Fig. 4.2: User Interface of the Coordination Tool

4.1.2 Conferee Identification

Conferees are identified with a RTPv2 [18] CNAME type handles. Vat, vic and web uses the following convention to identify a conferee and we adopt this convention to be used also in the coordination tool.

> user@host.domain.name or host.domain.name

when the host is a multi-user system the login id of the user with the sign is prefixed to the fully qualified domain name of the conference host. The host name can be replaced with the IP address of the host when the DNS inverse zone is not available.

4.2 Policy Specification

As described in section 3.2.3, each policies can be specified as a form of voting.

> Poll(label, initiator, rule, query-dialogue, notify-dialogue, reject-dialogue, confbus-message, variable i, value value-i, variable j, value value-j, ...)

4.2.1 Syntax

The following syntax is used to specify policies. The policy description is written in two parts; one that declares shared variables, and one that declares any operation upon those variables.

`global V(`*variable*`)` *value*

...

`ct_pack`

label

initiator

voting-condition

query-dialog

notify-dialog

pass-code

reject-dialog

fail-code

varlist-number

{

variable1

value1

...

}

Each of the items written in italics are further explained as follows. The language syntax used to describe the policies are that of Tcl[16]. When there is no explicit notation given, the Tcl notation is assumed.

label String to be bound with the menu item under the policy menu of the coordination tool.

initiator Variable*List of variables can also be used, but the coordination tool prototype doesn't implement this feature yet* to designate who can initiate the operation. There are predefined variables that can be used as the *initiator* string. `$myself`, `$creator` are examples of such variables. The following section describes more details of predefined variables. Variables defined in the first section of the specification can also be used.

voting-condition Predefined keywords or participants from which positive response should be collected to pass the vote. `pass-always`, `majority`, and `unanimous` are the list of predefind keywords.

query-dialogue Query string to be used in the dialog asking for a confirmation.

notify-dialogue Notify string to be used in the dialog when the vote is passed.

reject-dialogue Notify string to be used in the dialog when the vote is rejected.

pass-code Tcl code to be invoked when the poll passes. This is usually used to control media applications via messages over the conference bus.

fail-code Tck code to be invoked when the poll fails. This is usually used to handle failure recoveries.

varlist-number Number of variables passed in the variable list.

4.2.2 Predefined Variables

The following is a list of predefined variables. More variables can be defined as required as we gain more experience using the coordination tool.

$myself Same as $cname variable that is used to identify the conferee

$creator Creator of the conference

$title Session Title

$audio Format of the audio

$video Format of the video

4.2.3 Initialization

Distribution of initial policy specifications and initial variables is made available from the Session Description Protocol version 2 [8]. Coordination tool is launched from the session directory with initial variables passed in the arguments list.

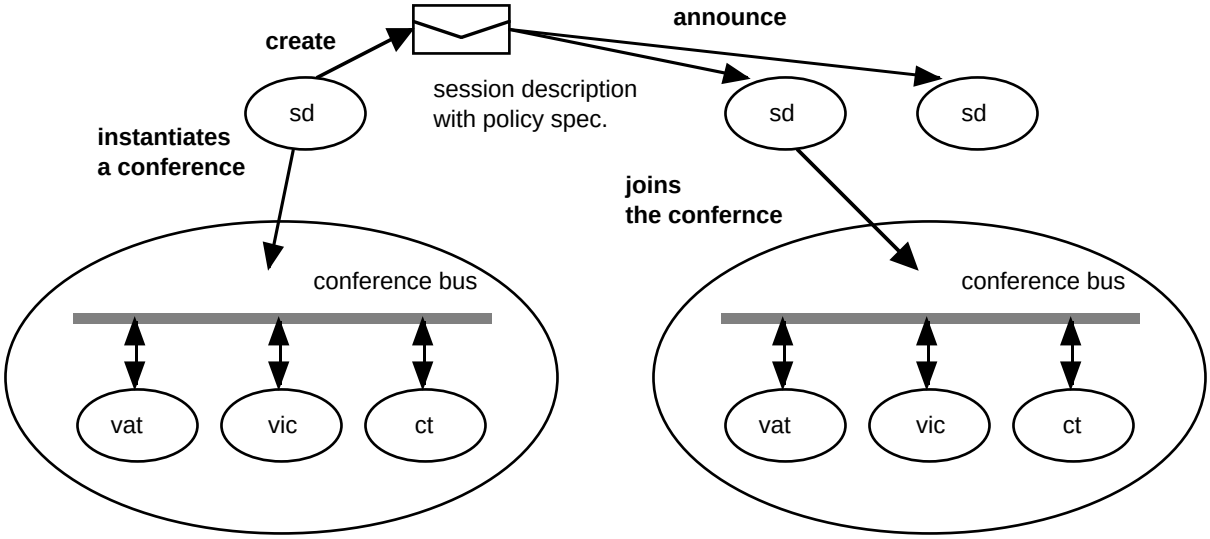

Fig. 4.3: The Runtime Model

4.3 Agreement Protocol

In a nutshell, the agreement protocol used in the coordination tool works in a simple manner as shown in the figure 4.4.

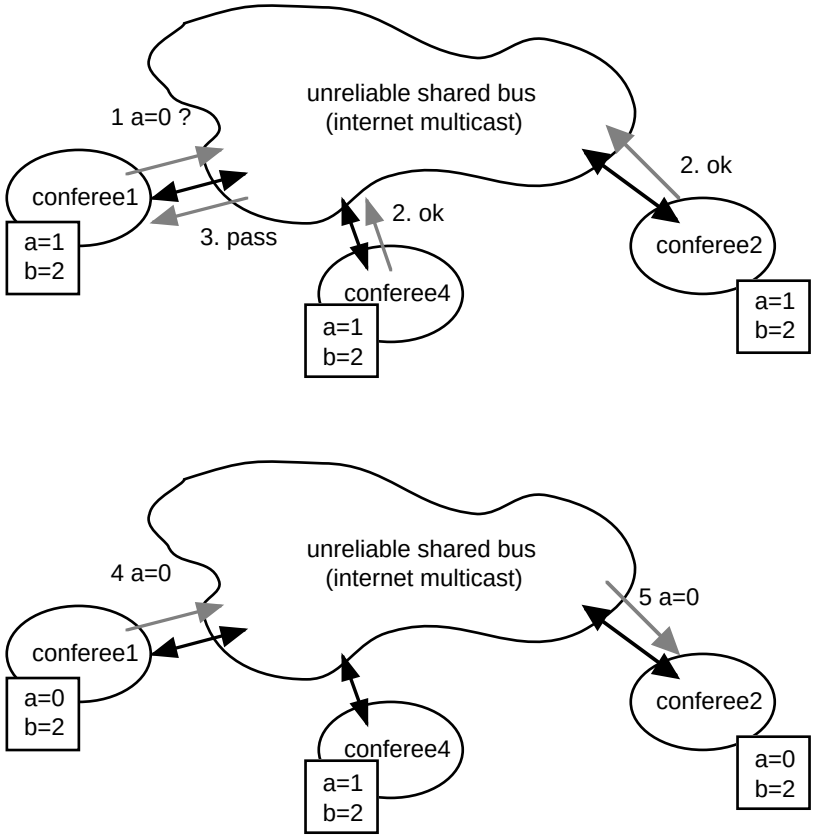

Fig. 4.4: Agreement Protocol in a Nutshell

The initiator sends a poll message to the multicast channel to which all coordination tools are subscribed. When other conferees' receive the poll message, they (when they find they are addressed to themselves) send response messages back to the initiator via the multicast channel. The initiator decodes each response messages for a certain period of time, updates the varaible when the voting condition is passed. state messages are sent

periodically to keep consistencies among the conferees.

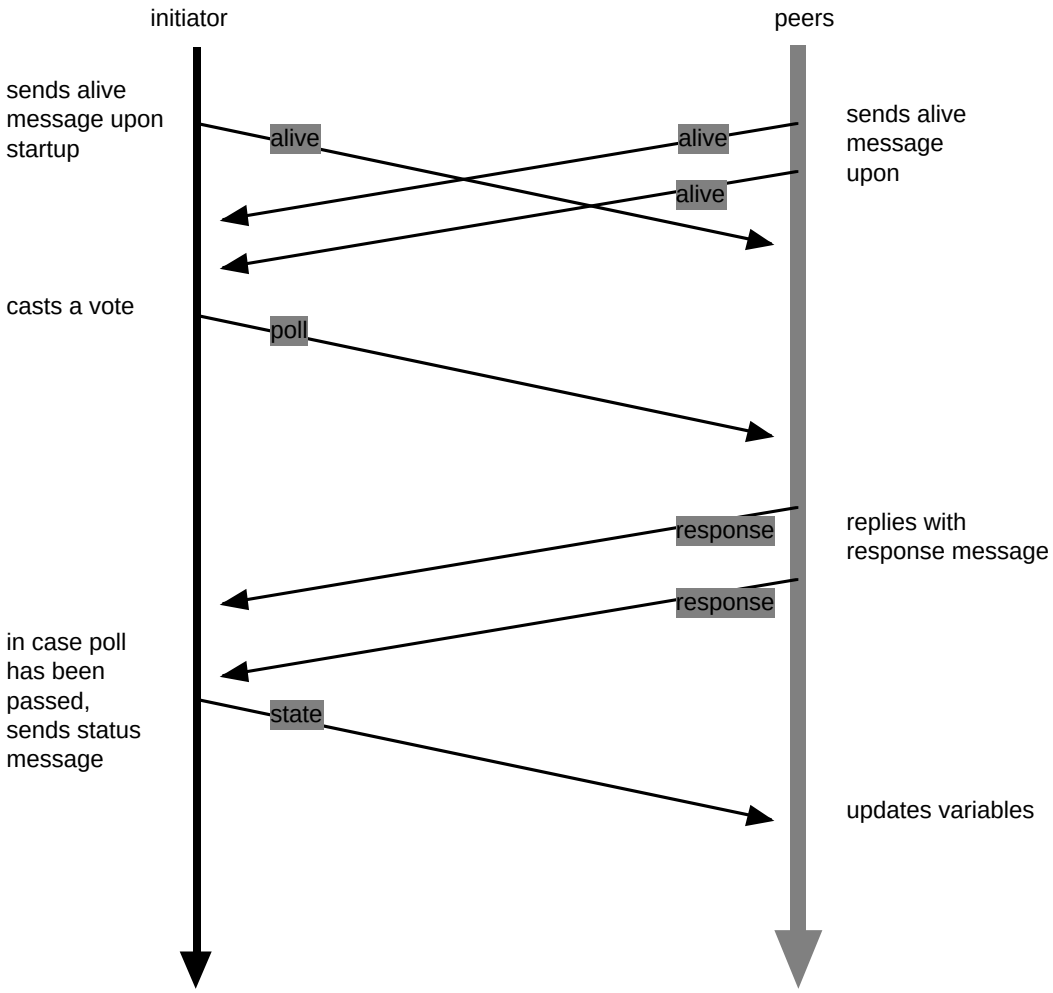

Fig. 4.5: Agreement Protocol Messages

alive messages are periodically sent by all coordination tools participating in the conference to identify conferees' who is alive. The frequency each alive and state messages are sent is determined by the number of participants to keep the bandwidth bound on a constant number. The delay in learning the new state also increases with the number of members since the overhead is kept constant by reducing the update frequency as membership increases.

This is similar to the algorithm used in the Internet teleconferencing tools vat and nv to maintain lists of members of the conference. Each new member is apprised of the current state by the incoming state messages. The lack of an initial state status exchange allows this mechanism to efficiently support an open membership policy (anybody who wants to can join), since then membership is merely announced by beginning to send state messages; the new member need not contact an old member to be initiated into the group.

Message Format

The protocol messages are formatted in plain ascii texts. There are four message types defined for the agreement protocol.

poll *src* poll *id dst query-dialog*

response *src* response *id dst response*

status *src* status *var-num var-list pass-code*

alive *src* alive *name*

Items written in italics above have the following meanings

src Identifier of the source

dst Identifier of the destination

id Unique identification number given to the poll message generated by the source. The tuple *src*, *id* uniquely identifies a given poll.

query-dialog The dialogue string to be used when prompting for a user to poll.

response Response to a poll. Either of `yes`, `no`, or `abstain`.

var-num Number of varibles in the *var-list*.

var-list List of variables and values. { *variable1 value1* $\cdots$ }

name Human readable form of the conferee's name

pass-code The Tcl code to be executed when the variable is changed.

4.4 Conference Bus Messages

Media Applications such as vat, vic, and wb are already designed to support conference buses. The user interface part of these applications are built with Tcl so that new message types can be easily handled by the applications. The only message currently being used is focus message which is used to support the voice-switched window feature in vat and vic.

Whenever a new message type is defined, it can be declared as a `cb_dispatch` handle, and the handler function can be bound with the dispatch handle. For example, the mute message in vat can be implemented as follows.

```
# conference bus API
# $cb send "mute $cname"

set cb_dispatch(mute) mute_someone

proc mute_someone cname {
audio $cname mute
}
```

Since many of the MBone applications currently use Tcl to control user interface parts, messages in the conference bus are written in Tcl.

Chapter 5

APPLICATION

The coordination tool described in the previous chapter can be used to support various types of conference policies. The followings show some of the possible use of the coordination tool in action.

Explicitly chaired conference

In an explicit chaired conference, a chairperson decides when someone can send audio and video. There are three policy descriptions, i.e. *Request Floor*, *Release Floor*, and *Revoke Floor*.

The variables used in the policy desciption are declared first. The first policy defines "Any conferee needs explicit permission of the chairperson before she/he can talk." The second one defines "Speaker returns floor when she/he finishes to talk." "Floor can be revoked by the chairperson" is what it means by the third one.

```
global V(chair) whchoi@cosmos.kaist.ac.kr
global V(speaker) ""

ct_pack
    "Request Floor" \
```

```
    $V(myself) \
    $V(chair) \
    "Can I speak next time?" \
    "You can speak now" \
    { confbus "mute all" confbus "unmute $V(speaker)" } \
    "You are not allowed talk right now" \
    { } \
    1 \
    { speaker $myself }

ct_pack
    "Release Floor" \
    $V(speaker) \
    pass-always \
    "" \
    "" \
    { confbus "mute $V(speaker)" } \
    "" \
    { } \
    1 \
    { speaker "unknown" }

ct_pack
    "Revoke Floor" \
    $V(chair) \
    pass-always \
    { confbus "mute $V(speaker)" } \
```

```
    "" \
    "" \
    "" \
    1 \
    { speaker "unknown" } \
```

Token-passing conference

In the token passing conference, the potential speaker asks for floor to the current token holder. This is very similar to the previous example, but there is no conference moderator at all. The policy description has a failure recovery code to handle when there is no current speaker defined.

```
global V(speaker)


ct_pack
    "Request Floor" \
    $V(myself) \
    $V(speaker) \
    "Can I speak next time?" \
    "You can speak now" \
    { confbus "mute all" confbus "unmute $V(speaker)" } \
    "You are not allowed talk right now" \
    { if [expr $V{speaker} = "unknown"] set $V(speaker) $myself reenter } \
    1 \
    { speaker $myself } \
```

Changing the audio format

Assume that unanimous agreement is required to change the audio format. The following example shows a policy specification to change the current audio format to a low bandwidth format.

```
ct_pack
    "Change Audio to Low" \
    $V(myself) \
    unanimous \
    "Can I speak next time?" \
    "Audio format changed to low quality GSM" \
    { confbus "select_format gsm 4" } \
    "Audio format could be changed" \
    { } \
    1 \
    { audio "gsm 4" }
```

Chapter 6

ANALYSIS

Before we discuss benefits and drawbacks of the conference control model proposed in the paper, we briefly give an overview of the previous chapters.

By investigating the state-of-the-art works in the area of conference control for light-weight sessions, we had the following requirements for the conference control model.

- Existing applications be used without much modifications
- Retain scaling properties of media applications
- Support flexible policies

To meet these requirements, a conference control model has been proposed. The model had a coordination tool in the central part of the architecture which used an agreement protocol to manage shared states among the conferees. The agreement protocol assumed an unreliable shared-bus communiation model as it is in the Internet multicast communications. The coordination tool could colaborate with media applications via the conference bus abstraction.

A prototype of the coordination tool has been implemented in a small Tcl-DP code which is around 1000 lines long. This could be made possible

by high-level communiactions and string manipulation functions provided in Tcl-DP.

We now discuss benefits and drawbacks of the conference control model proposed in the paper. The conference control model is based on the light-weight session model of the Internet multicast conferencing. Thus it inherits many of the properties of the MBone applications. The conference control model is compared with traditional MBone applications, the Conference Control Channel Protocol, and ITU T.124 recommendation in terms of scalability, reliability, flexible policies, the level of control provided, availability of applications.

	Scalibility	Reliability	Flexible Policies	Level of Control	Availibility
Traditional MBone	○	△	×	×	○
CCCP	○	○	–	–	×
Coordination Tool	○	△	△	△	△
ITU T.124	×	○	×	○	○

Fig. 6.1: Comparison between Conference Control Models

Scalability

Traditional MBone tools lacked any meaningful conference control features, but they are most scalable due to their communications model, IP multicast. The coordination tool works with existing MBone applications without any modifications in the communication architecture, so media applications

themselves retain their scalability. The agreement protocol underlying in the coordination tool is at least as scalable as the media applications themselves since the protocol is based on the multicast communication model and is implemented in a similar mechanism that is found in Internet multicasting conference applications.

CCCP is as scalable as MBone applications when the transport reliability condition is relaxed. ITU T.124 Recommendation is based on the centralized Multipoint Communication Unit(MCU). MCU's can be cascaded to support broader coverage, but it is not as scalible as Internet multicast since T.124 assumes point-to-point communications model.

Reliability

Reliability can always be provided by sacrificing scalability. The coordination tool doesn't use any reliable transport protocol. There is always non zero probability that all messages be lost in unreliable multicast communications. However, the agreement protocol guarantees weak eventual consistency[19] in shared states. CCCP has an reliability option, and T.124 is based on a reliable transport protocol.

Flexible Policies

Policies are represented in a simple script in the coordination tool. The script can be written by anybody who wants to implement a different policy, and can be incorporated into the coordination tool in run time. However the policies themselves cannot be changed durinng the course of the conference in the conference control model proposed in this paper.

The conference control architecture in CCCP allows floor control and ses-

sion control applications be modularized and replaced at will. But CCCP leave it to designers of control applications how to support flexible policies. T.124 lends the policy implementations into the each conferencing applications. Policies cannot be changed unless the application is designed to support flexible policies.

Level of Control

The conference control model proposed in this paper is based on the conference bus where media applications and the coordination tool can speak with each other. New messages can be defined and implemented in applications to provide more control features. On the other hand, these could be limitation of the conference control model proposed in the paper. The conference control model lacks any well-defined set of application control interface.

Comparison with CCCP in this perspective is not applicable since CCCP leaves this also to the hands of the control application designers. In the case of T.124, applications can be built to provide more control features than that is defined in T.124, but this is not likely since T.124 would be the least and the most common denominator in most cases.

Availability of Applications

MBone applications are available now, and they can be used in controlled conferences without any modifications with the help of the coordination tool. T.124 based products are already available in market places while there is no known control applications based on CCCP as of this writing. The coordination tool need to be fully integrated with session directory applications and this should be resolved with further works.

Chapter 7

CONCLUSION

This paper has presented a conference control model for Internet multicasting applications. The model had on two kinds of communications involved; the conference bus for collaborations among media applications and the coordination tool, and communications among the conferees' coordination tools where they use an agreement protocol to assure state consistencies. A coordination tool has been built to support policies in existing applications and various applications of the coordination tool has been demonstrated.

Two key changes that are made possible by the model are that; The model does not rely on cooperation among all the remote participants in a session. A misbehaving participant cannot cause problems because it will be muted by all participants that follow the protocol. Another benefit of the model is that the conference policy can be changed at will since the model has been designed to carefully separated policies from the mechanisms that implement the policies.

The model can be incorporated with session directory tools and session description protocols to keep the directory information keep up-to-date so that late comers can join the on-going conference without any problems. The coordination tool might also be extended to support reliable communications sacrificing the scalability to support tightly coupled conferences.

Appendix A

TERMINOLOGIES

This section describe the terminologies used throughout the paper. Many of the literatures in the area of conference control use these terms in different ways, and it is necessary to provide definitions of the terms used to avoid potential confusions. These terminologies are adopted from the discussions in the IETF MMUSIC working group[5].

General

Conference A logical abstraction among multiple participants for multimedia real-time communication. A conference consists of a control session, related media associations, and conference policies.

Conferee A participant in a conference.

Session An association of members for control; for instance, to control a conference with multiple conferees.

Media Association Encapsulation of the transport (point-to-point or multi-point) in a single medium.

Conference Policies Rules regarding the style of interaction for a conference.

Control

(Multiparty Multimedia) Session Control Protocol The protocol used for session control, the management and coordination of multiple sessions, and their multiple members in multiple media.

Distributed Control A control model where control functions are distributed among session managers.

Centralized Control A control model where control functions are the responsibility of a centralized agent.

Tight Control A session style in which state is actively shared among participants and that aims to keep state consistent among participants.

Loose Control A sessions style in which state information is passively shared among participants. In the extreme, no state sharing is performed.

System Components

Conference Session Manager A software entity that resides at each conferee's end system to coordinate the initiation, maintenance and interactions of sessions. A communication endpoint for the session control protocol.

Media Agent A software entity that handles media-specific functions such as encoding, compression and transport packetization that are used by conferences. Media in a conference might include audio, video, graphics and text.

Resource Manager A software entity that manages the media agents on a workstation. It understands the static end-system descriptions (hard-

ware and software capabilities), as well as the negotiated per-session preferences.

Conference Directory Service A directory that provides user network addresses, conference IDs and addresses, the conference begin time, conference topic, etc.

Conference Scheduling The advertisement of a conference's start time with a session directory service.

Policies

Initiator A conferee that initiates a conference.

Chair A designated conferee having more authority than other conferees in the conference. For example, the chair might decide the policy on late joins, media floor control, interaction style, etc.

Receiver A conferee that receives session data.

Sender A conferee that transmits/sends session data.

Passive Participant A conferee that only acts as a receiver

Active Participant A conferee that acts both as a sender and receiver

Reflector Participant An entity that relays data between conferees, acting as a go-between.

Floor Control Coordinated control over who may or may not send and/or receive data.

Connectivity Style The interconnectivity of conferees (e.g. 1-to-N, N-to-N, M-to-N) in either the control or data realm.

Access Control The accessibility of a session to potential conferees.

Interaction Policies The model and rules used by conferees to interact with one another in a conference.

References

[1] S. Casner. First IETF Internet Audiocast. *ACM SIGCOMM Computer Communications Review*, July 1992.

[2] T. Crowley *et al.* MMConf: An infrastructure for building Shared Multimedia Applications. In *Proceedings of CSCW'90*, Los Angeles, USA, October 1990.

[3] M. Arango *et al.* The Touring Machine system. *Communications of the ACM*, 36(1), January 1993.

[4] R. Frederick. *nv UNIX Manual Pages*. Xerox Palo Alto Research Center, Palo Alto, USA.

[5] MMUSIC Working Group. The glossary taken from slides presented at the MMUSIC meeting. The 27th Internet Enginnering Task Force Meeting, July 1993.

[6] M. Handley. Minutes of the Multiparty Multimedia Session Control Working Group. The 31st Internet Enginnering Task Force Meeting, July 1995.

[7] M. Handley, J. Crowcroft, and C. Bormann. The internet multimedia conferencing architecture. Technical report, Internet-Draft, February 1996.

[8] M. Handley and V. Jacobson. SDP: Session Description Protocol (draft 2.1). Technical report, Internet Draft, February 1996.

[9] M. Handley, P. Kirstein, and A. Sasse. Multimedia Integrated Conferencing for European Researchers (MICE): Piloting Acticities and the

Conference Management and Multiplexing Center. *Computer Networks and ISDN Systems*, 26, November 1993.

[10] M. Handley, I. Wakeman, and J. Crowcroft. The Conference Control Channel Protocol (CCCP): A scalable base for building conference control applications. In *Proceedings of ACM SIGCOMM'95*, Boston, USA, August 1995.

[11] V. Jacobson and S. McCanne. *Using the LBL Network Whiteboard.* Lawrence Berkeley Laboratory, Berkeley, USA.

[12] V. Jacobson and S. McCanne. *vat UNIX Manual Pages.* Lawrence Berkeley Laboratory, Berkeley, USA.

[13] V. Jacobson, S. McCanne, and S. Floyd. A Conferencing Architecture for Light-weight Sessions. Technical report, Lawrence Berkeley Laboratory, November 1993.

[14] M. Macedonia and D. Brutzman. MBONE provides Audio and Video across the Internet. *IEEE Computer*, 27(4), April 1994.

[15] S. McCanne and V. Jacobson. vic: A Flexible Framework for Packet Video. In *Proceedings of ACM Multimedia'95*, San Francisco, USA, November 1995.

[16] J. Ousterhout. *Tcl and the Tk Toolkit.* Addison-Wesley, 1994.

[17] E. Schooler, R. Lang, and M. Handley. Charter of the Multiparty Multimedia Session Control Working Group. Internet Engineering Task Force Working Group.

[18] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. Technical report, RFC 1889, January 1996.

[19] S. Shenker, A. Weinrib, and E. Schooler. Managing Shared Ephemeral Teleconferencing State: Policy and Mechanism. Technical report, Internet-Draft, July 1993.

[20] T. Turletti. *INRIA Video Conferencing System(ivs).* Institut National de Recherche en Informatique et an Automatique.

[21] International Telecommunication Union. *ITU Recommendation T.124 – Generic Conference Control.*

[22] H. Vin *et al.* Multimedia Conferencing in the Etherphone Environment. *IEEE Computer*, 24(10), October 1991.

[23] L. Zhang, S. Deering, D. Estrin, S. Schenker, and D. Zappala. RSVP: A New Resource ReSerVation Protocol. *IEEE Network*, September 1993.