A User-Centric Network Communication Broker for Multimedia Collaborative Computing

Chi Zhang[1], S. Masoud Sadjadi[2], Weixiang Sun[3], Raju Rangaswami[2], Yi Deng[4]

[1]*Juniper Networks*
Sunnyvale, CA, USA
chizhang@juniper.net

[2]*School of Computing and Information Sciences*
Florida International University
Miami, FL, USA
{sadjadi, raju, deng}@cis.fiu.edu

[3]*Amazon*
Seattle, WA, USA
weixiang@amazon.com

[4]*College of Computing and Informatics*
University of North Carolina at Charlotte
Charlotte, NC, USA
Yi.Deng@uncc.edu

Abstract

The development of collaborative multimedia applications today follows a vertical development approach, where each application is built on top of low-level network abstractions such as the socket interface. This stovepipe development process is a major inhibitor that drives up the cost of development and slows down the innovation pace of new generations of communication applications. In this paper, we propose a network communication broker (NCB) that provides a unified higher-level abstraction for the class of multimedia collaborative applications. We demonstrate how NCB encapsulates the complexity of network-level communication control and media delivery, and expedites the development of applications with various communication logics. We investigate the minimum necessary requirements for the NCB abstraction. We identify that the concept of user-level sessions involving multiple parties and multiple media, is critical to designing a reusable NCB to facilitate next-generation multimedia communications. Furthermore, the internal design of NCB decouples the user-level sessions from network-level sessions, so that the NCB framework can accommodate heterogeneous networks, and applications can be easily ported to new network environments. In addition, we demonstrate how the extensible and self-managing design of NCB supports dynamic adaptation in response to changes in network conditions and user requirements.

1. Introduction

The convergence of various multimedia communications including voice, video and data over IP networks during the past decade has resulted in the emergence of a wide range of collaborative applications including Scientific Collaboration, Video Conferencing, Voice over IP (VoIP), and Instant Messaging, among others. These collaborative applications have the potential to dramatically impact our everyday life. However, the fast pace growth of innovations has been restrained by the stovepipe approach currently employed in application development.

Today, the development of *domain-specific* collaborative applications is both time-consuming and error-prone because the low-level communication services provided by the existing systems and networks are primitive and often heterogeneous. Multimedia collaborative applications are typically built on top of low-level network abstractions such as TCP/UDP socket, SIP (Session Initiation Protocol) and RTP (Real-time Transport Protocol) APIs. Further, the underlying network configurations can also vary significantly which can reduce portability within applications developed using a vertical stovepipe approach.

What is lacking is a systematic approach to design and development across high-level collaborative applications. In [6], we introduced Communication Virtual Machine (CVM) that represents a paradigm shift in how a collaborative application is conceived and delivered. In CVM, general-purpose or domain-specific communication needs are specified in a model, called communication schema, independent of device types and underlying network configuration. Such a model is instantiated, negotiated, synthesized and executed, by a fully automated process, to satisfy the users' communication needs. Under this model, a communication modeling language is proposed to provide an intuitive graphic form for users to model declaratively their communication requirements. A synthesis engine is further responsible for automatically synthesizing user communication sessions. This model-driven communication is supported by the CVM layered architecture. The layered architecture provides separation of major concerns such as modeling application-dependent collaboration logic, automatic generation of scripts to drive the collaboration logic, and the application-independent basic communication service reusable by various applications (i.e. network communication broker to interface with the underlying network infrastructure).

As the basic application-independent communication service that actually delivers the communication is fundamental not only to the CVM architecture but also to the development of any collaborative application, we focus on the abstraction, internal architecture, and implementation of this layer in this paper, based on our early primary work in [34]. We propose Network Communication Broker (NCB), a user-centric middleware that encapsulates the networking complexity and heterogeneity of basic multimedia and multi-party communication for upper-layer collaborative applications. As illustrated in Figure 1, NCB provides a unified user-centric communication service to diverse collaborative communication applications ranging from a simple phone call and video conferencing to specialized communication applications like scientific collaboration, disaster management, and telemedicine. Under this unified high-level abstraction, internally NCB coordinates the underlying heterogeneous network infrastructure, systems and libraries to ensure that basic communication tasks are carried out smoothly. The key innovation of the NCB concept is a horizontal abstraction that separates and isolates the complexities of network-level communication control and media delivery from the diversity of application-dependent collaboration logic.

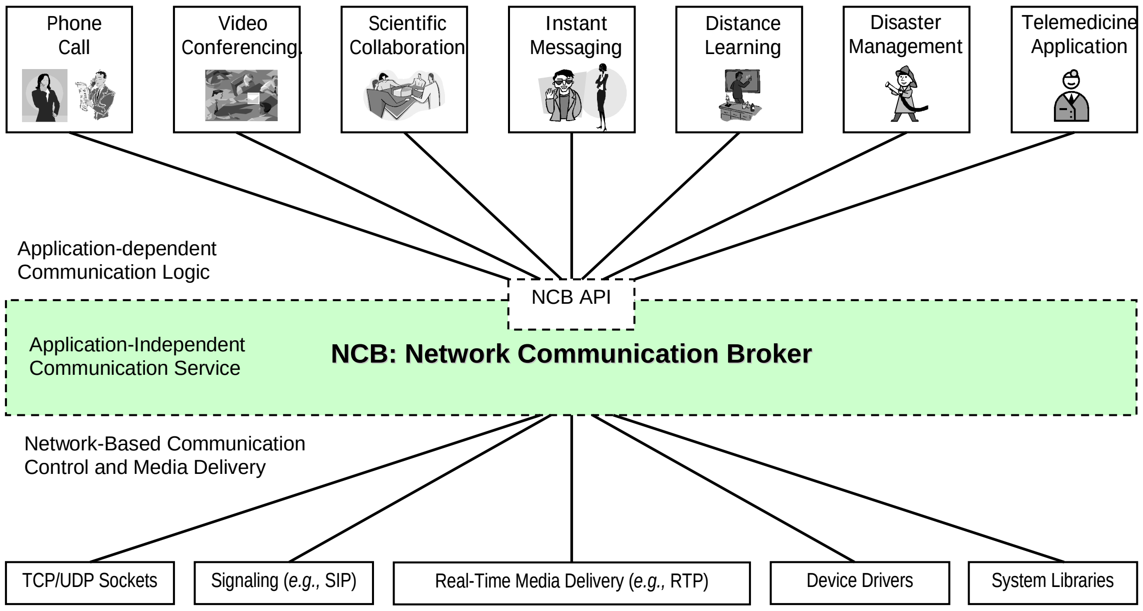

Phone Call
Video Conferencing
Scientific Collaboration
Instant Messaging
Distance Learning
Disaster Management
Telemedicine Application
Application-dependent Communication Logic
NCB API
Application-Independent Communication Service
NCB: Network Communication Broker
Network-Based Communication Control and Media Delivery
TCP/UDP Sockets
Signaling (e.g., SIP)
Real-Time Media Delivery (e.g., RTP)
Device Drivers
System Libraries

Figure 1: NCB separates network communication complexity from application collaboration logic.

Since the NCB provides a service of user-centric multimedia collaboration, we identify the scope of its encapsulation more specifically to multi-party multimedia telecommunication. We summarize the values of this unified NCB encapsulation as follows:

- The unified abstraction by itself is generic enough to provide basic user-centric communication services reusable by a wide variety of collaborative applications.
- Applications, developed based on NCB, are transparent to the details of underlying network protocols and infrastructure. The high-level application-dependent collaboration logic therefore becomes relatively simple to build.
- NCB hides network heterogeneity from the applications so that applications can be easily ported to new network environments.

In this paper, we investigate the minimum necessary requirements for the NCB to be able to support complex collaboration logic involving multiple parties and multiple media and to provide an interface that exemplifies this abstraction. In terms of approaches, the paper has made two novel contributions. We first identified that the concept of *user-level sessions* (vs. network-level sessions adopted by the existing protocols) is critical to designing a flexible NCB interface to support various next-generation collaborative communications. The internal design of NCB separates the user-level sessions and network-level sessions, such that the NCB framework can accommodate heterogeneous networks.

Furthermore, we demonstrate how the extensible and self-managing design of NCB supports dynamic adaptation in response to changes in network conditions and user requirements. To be more specific, NCB contains a self-management module that can conduct self-optimization to autonomously adapt media-delivery to the changing network conditions (e.g. bandwidth). Moreover, the adaptive behavior of the middleware can be defined according to the preferences of users or applications, specified as high-level policies.

The rest of this paper is organized as follows. In Section 2, we identify the set of requirements for the NCB abstraction and present a minimal interface for NCB that reflects these requirements. In Section 3, we overview NCB's internal architecture and design. In Section 4, we introduce the prototype implementation of NCB in Java, as well as experiments and findings. Section 5 presents related work and Section 6 provides some concluding remarks.

2. NCB Abstraction and Interface

In this section, first, we identify the set of necessary requirements from the perspective of next-generation collaborative applications. We then present an interface that reflects the requirements.

2.1 NCB Abstraction Requirements

In contrast to traditional telephone networks, where end devices are “dumb” and all the complicated communication functions are controlled by servers/switches, in IP networks designed based on the end-to-end argument, applications on end-hosts can deliver sophisticated communication services. Finding the appropriate level of NCB abstraction for applications in IP networks is non-trivial. An abstraction that is too high-level can reduce the flexibility of the applications. On the other hand, an abstraction that is too low-level, can significantly complicate the task of the developer, and reduce portability. For instance, an abstraction of sockets on top of transport-layer protocols is too primitive, as it does not hide the complicacy of signaling protocols etc., which is crucial to multi-party multimedia communication. Furthermore, as discussed in section 3, these protocols’ functioning depends on the network infrastructure and conditions, which should be hidden to the design of portable applications. As another example, an abstraction encapsulating telemedicine is too high-level for NCB, as application-specific logic of telemedicine cannot fit into, say, distance learning.

We examined a spectrum of collaborative applications (including the ones listed in Figure 1.) and various aspects of their development. We extracted the communication issues common to all these applications, and especially those issues depending on the underlying network infrastructures and conditions that will affect application portability. As a result, we elaborate below on the key aspects of an abstraction of high-level communication services that can satisfy the needs of next generation collaborative applications.

First, the NCB abstraction must provide a registration mechanism for all users that allows an individual user as well as the system to locate users for establishing communication. The NCB interface must also support the basic presence functionality so that a user can retrieve and modify his/her current contact-list of individuals that he/she may desire to communicate with.

Second, the abstraction should support the basic concept of a *user session*. We define a user session of NCB as a communication process that involves a number of participants, who can be added or removed dynamically. A user session thus represents a “multicast communication space”, within which each participant can send media to all the other session participants. Within each user session, the participant should be able to deliver various media on demand, such as sending a document in the middle of a voice communication. Further, the NCB abstraction must be capable of supporting multiple user sessions simultaneously, which is necessary for sophisticated collaborative applications such as disaster management. In response to a disaster, an administrator may initiate several user sessions from an end host to different groups, since different groups may have different communication topics, media types (e.g., voice communication in one user session and text chat in another), priorities and levels of secrecy. As another example, in a distance learning application, while all students participate in a user session of lecturing, one student may establish a private session to another student, asking for a document.

Third, the abstraction must comprehensively encapsulate the details of the networking infrastructure. The abstraction should ensure that applications can be ported easily to different end-host hardware and different underlying network infrastructure.

Fourth, the interface should also provide a mechanism to expose certain low-level network and system events (e.g. network outage) to applications, since under certain circumstances, the upper-layer application may desire to be notified of the communication states, so that appropriate decisions can be made based on its application-dependant communication logic. This capability essentially enables development of context-aware applications on top of NCB.

Finally, the interface must provide users the capability to specify high-level self-management policies to be used as guidance inside the NCB for controlling how media is exchanged and delivered. In some applications, the user may desire to dynamically control the behavior of NCB in session control and media delivery, according to his/her preferences, by specifying high-level policies. The NCB abstraction must be flexible enough to support such requirements.

2.2 NCB Initialization and Presence Interface

The NCB initialization and presence interface allows an application to register a user for communication purposes with a signaling server. Table 1 presents the proposed interface.

Interface	Description
`void launch()`	Configures NCB based on a configuration file
`void shutdown()`	Do cleanup for NCB
`boolean login(String realm, String user, String passwd)`	Login into signaling server
`boolean logout()`	Logout from signaling server
`boolean add/removeContact(String name, String identifier)`	Add/Remove a new contact to/from the contact list

Table 1: NCB Initialization and Presence Interface.

2.3 NCB Session Interface

The session interface provides an application with the ability to create multiple independent user-level communication sessions. Each user session can be configured to allow multiple participants and various media delivery on demand. The session interface is presented in Table 2.

Interface	Description
`int createSession(String comments)`	Create new communication session
`boolean destroySession(int sid)`	Destroy an existing session
`public boolean add/remParty(int sid, ArrayList parties)`	Add or remove new parties to a session
`boolean add/remMedia(int sid, String media_type, String media_location)`	Add or remove a new media to a session
`boolean suspend/resumeMedia(int sid , String media_type,String media_location, String direction)`	Suspend or resume the data transmission for a media

Table 2: NCB Session Interface.

A session ID is returned by the NCB whenever a new session is created with the `createSession` call from the upper layer. The session ID is then used by the application to uniquely identify a user session maintained within the local NCB, in the subsequent calls to add/remove participants and media into/from the user session. Each user session may involve several types of media. Each media delivery within a session is uniquely identified by the URI (Universal Resource Identifiers) of the media. Diverse media types are supported (*e.g.*, real-time audio/video, instant messages, and files). In addition, NCB supports both two-way (*e.g.*, voice conversation) and one-way media transfer (*e.g.*, file transfer and distance learning) with different media formats, as show by the `direction` parameter.

2.4 NCB Callback Interface

The NCB callback interface in Table 3 provides a mechanism by which the NCB can notify the application of specific events of networks, sessions, participants, and media, which are used by the application in a custom fashion. The callback interface enhances NCB flexibility for different application logic. As an example, when a "Ringing-tone" signaling message is received by the caller, the upper layer may select a presentation (e.g., a flashing icon) rather than an audio ring on the speaker. This could be useful when the caller concurrently initiates several sessions in disaster management, or connect to several participants in a session of lecturing.

Interface	**Description**
`void networkFailure(String nwFailure)`	Notification of network failure
`void contactStatus(String user, int status)`	Report the presence of a contact
`void sessionStatus(int sid, String status)`	Report the status of a session (open, close etc.)
`void partyStatus(int sid, String user, int status)`	Report the status of a participant (busy-tone, ring-tone, join, etc.)
`void mediaStatus(int sid, String media_type, String media_URI, int status)`	Report the status of a media in a session

Table 3: NCB Callback Interface.

2.5 NCB Self-Management Interface

NCB internally conducts self-optimization to autonomously adapt media-delivery to the changing network conditions. The NCB self-management [10] interface allows upper-layer applications to customize (by specifying high-level policies as guidance) NCB adaptive behaviors under specific network and system conditions, based on user or application preferences. The interface provides two operations: `getPolicyStatus` and `applyPolicy`, presented in Table 4.

Interface	**Description**
`String getPolicyStatus()`	Get the current policy status of the NCB including the parameters related to the policy in effect and the policy itself in XML format
`int applyPolicy(String`	Apply the self-management policy specified by the

xmlString)	xmlString

Table 4: NCB Self-Management Interface.

We developed an XML Schema to be able to specify high-level policies in XML documents. At a high-level, a NCB self-management policy can be interpreted as an "*if* <condition> *then* <action>" construct. An example of a self-optimization (a sub-category of self-management) policy in XML that is currently supported by the NCB is shown in Figure 2. The policy is specific to the session with ID # 24 and it dictates that when NCB detects a low bandwidth condition, it should increase the video compression and vice-versa to maintain a steady frame-rate. A specific experiment showing the following policy in action and providing further details is presented in Section 4. Examples of other high-level self-management policies can be found in section 3.2.

```
<session sessionID="24">
    <connectionConstraint
     condition="networkBandwidthDecreasing"
     action="decreaseVideoResolution" />
    <connectionConstraint
     condition="networkBandwidthIncreasing"
     action="increaseVideoResolution" />
</session>
```

Figure 2. Example self-optimization policy specification.

We claim that the existing NCB abstraction is not only simple and easy to use, but also provides bare minimum middleware support for developing collaborative multimedia applications. NCB provides applications with the flexibility of establishing full-featured communication user sessions with controls over the number and identity of participants and the nature and timeliness of media exchanged during each session. This minimum interface can already support a variety of next-generation collaborative applications.

3. NCB Internal Architecture and Design

The challenge for NCB internal architecture is to identify a unified and extensible framework to accommodate (i) diverse media with different processing (ii) diverse protocols (iii) diverse communication features and requirements (iv) diverse network infrastructures, and (v) diverse traffic conditions. Below, in section 3.1, we give concrete examples for NCB design issues. Section 3.2 details a middleware framework to address the aforementioned design challenges. Section 3.3 discusses the design choices in the design space.

3.1 NCB Design Issues and Principles

NCB is a client-side middleware to be deployed on end hosts. Below the unified NCB abstraction, the NCB core translates a high-level communication task into a series of operations that control and coordinate the underlying networking facilities to deliver media to session participants. The NCB core is complex in that it coordinates both the control plane (*i.e.*, signaling protocols negotiating the communication) and the data plane (*i.e.*, transport protocols delivering media).

The introduction of NCB will not affect or require changes to the existing protocols and network infrastructures, in order not to re-invent the wheel of basic communication. The communication between peer NCBs may travel through various communications infrastructures, such as signaling servers and media gateways, and follows various protocol standards (*e.g.*, SIP [9] and RTP [15]). For example, there can be an existing SIP server that authenticates, processes and forwards signaling messages for user registration as well as parameter negotiation (*e.g.* media to be transmitted, encoding/decoding schemes, device capabilities and presence). The IP address of the signaling server and its service port for SIP signaling must be configured into the NCB during initialization through the interface listed in section 2.2. In addition, there may be a media gateway converting diverse audio/video encoding schemes from different NCBs. The multimedia gateway will further mix real-time audio/video signals for multi-party conferencing and then do multicasting. Without a mixer, each participant will send his/her audio to all the other participants in duplicated streams [19], each of which may have different SIP session ID and RTP session ID.

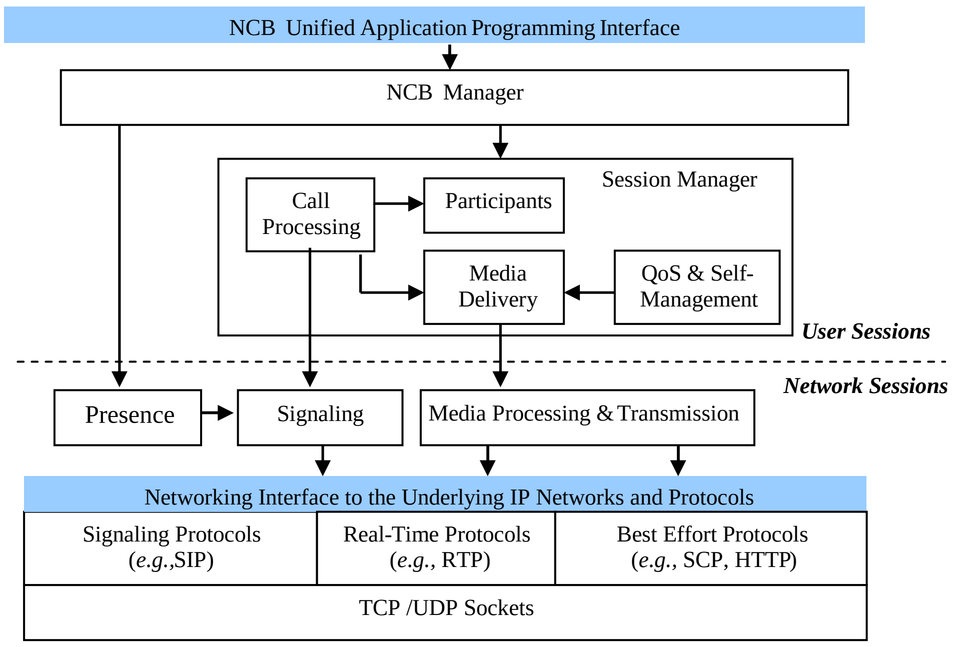
However, just like a socket number and its associated port that hide the communication details of packet transmission, the NCB user session (see section 2.1) encapsulates the complexity of multiparty, multimedia communication. For instance, the communication messages between different NCBs following standard networking protocols may have their own notions of low-level sessions or session IDs, and do not contain NCB session IDs. To encapsulate various *network sessions* within one *user session* for our user-centric middleware, NCB must internally maintain the mapping between the NCB session and the sessions of the underlying protocols. In the rest of the paper, the term “session” is used to denote a NCB user session, unless otherwise stated.

The NCB internal architecture must also have an extensible and reusable framework facilitating the integration of new communication functionality, new media types, and new networking primitives (*e.g.,* QoS), with different network configurations. The NCB is also designed to be self-optimizing, so that the middleware can automatically adapt to dynamic network conditions, such as available bandwidth, packet loss rate, and energy consumption.

3.2 NCB Internal Architecture

The internal architecture of NCB is outlined in Figure 3. We briefly describe and discuss each module as follows. In addition, a state chart showing a prototype that implements this framework can be found in section 4.

(a) NCB Manager**:** The NCB Manager is responsible for the initialization and the configuration of the NCB middleware. For example, it maintains the signaling server information. Also it is responsible for registering the user information (*e.g.* the current mobile IP address at which it can be reached for signaling messages). Upon receiving an application request for creating a new session (at the caller side), or a signaling message INVITE (at the callee side) from a remote user negotiating a new conversation, NCB Manager creates a new Session Manager (see below) to handle the new communication session. The NCB Manager maintains the list of Session Managers for all active sessions. In addition, it handles states relevant to all sessions that cannot be handled by individual Session Managers. For example, in case of multiple user sessions of voice communication, the NCB Manager can activate one voice session and mute all the other voice sessions. The application can control the active session through the `resumeMedia/suspendMedia` interface given in section 2.3, thus implementing the call-waiting service.


NCB Unified Application Programming Interface
NCB Manager
Session Manager
Call Processing
Participants
Media Delivery
QoS & Self-Management
User Sessions
Network Sessions
Presence
Signaling
Media Processing & Transmission
Networking Interface to the Underlying IP Networks and Protocols
Signaling Protocols (e.g.,SIP)
Real-Time Protocols (e.g., RTP)
Best Effort Protocols (e.g., SCP, HTTP)
TCP /UDP Sockets

(b) Session Manager**:** A session manager deals with a single user session. Since the states associated with a session include the call status, the participants, and the media transfer, this module further dispatches the tasks to the "Call Processing", "Session Participants", and "Media Delivery" sub-modules within the Session Manager.

The *Session Participants* module keeps the list of active participants of this session.

The *Call Processing* module controls, at the level of user sessions, the logic of a session. It converts high-level control actions (such as "`addParticipant`") of a user session to low-level signaling operations, based on the underlying signaling module, which actually carries out the basic signaling. When receiving a signaling message indicating that a new participant joins the session, the Call Processing module invokes the Participants module to update the participant list, and then reports the newly joined participant to the upper-layer application (through the `partyStatus` callback interface in Table 3).

The *Media Delivery* module manages, at the level of user sessions, the transfer of media in a session. It translates an "`addMedia`" call from the application into a number of internal operations. It first relies on the Call Processing module to negotiate transmission parameters (*e.g.* encoding/decoding schemes and QoS parameters) before the actual media transmission. It then controls on the *Media Processing and Transmission* module to actually transmit the media (see Figure 4). Some media, such as short messages, can be delivered within the signaling messages (*e.g.*, SIP Messages), and thus go through the Signaling module.

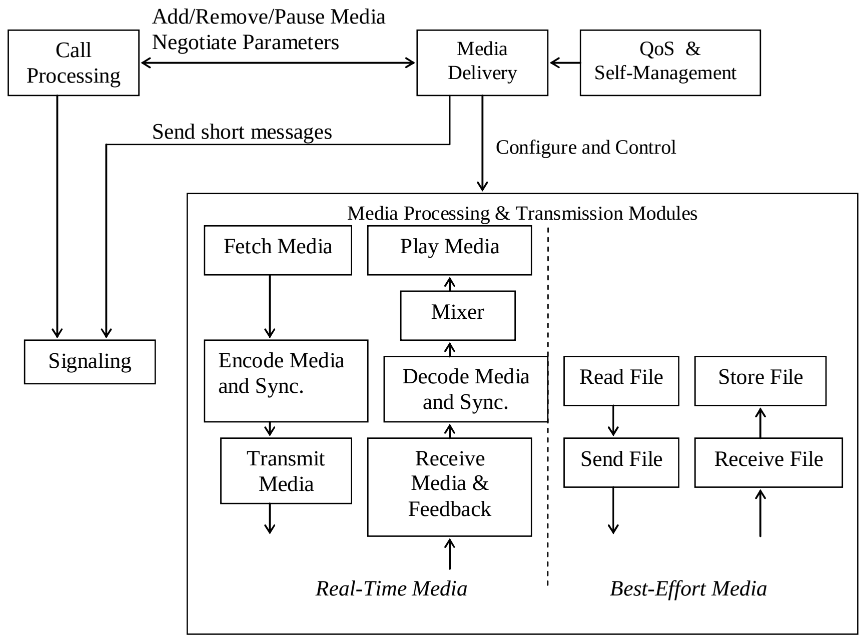


Figure 4. Signaling and Media Delivery.

(c) Media Processing and Transmission: The media will be pre-processed before transmission at the sender side, and will be post-processed and recovered at the receiver side. The processing and transmission/reception depend largely on the heterogeneous media types and network configurations (*e.g.* with or without conferencing mixer discussed above). The *Media Processing and Transmission* module maintains the supported media types and the corresponding encoding/decoding schemes, and carries out media processing and transmission. In contrast to the *Media Delivery* module, this module is fully unaware of the states of a user-level session.

As shown in Figure 4, the module has different processing paths for real-time media delivery and best-effort data delivery (*e.g.* documents). For best-effort data delivery, everything can be blindly transmitted as a file. In contrast, for multimedia content, encoding/decoding is necessary, as media transmission control depends on various the network conditions (*e.g.* available bandwidths) and media content, in order to maximize the user experience.

For example, with voice conferencing, the participants either rely on a conferencing server mixing the voices from different senders, or use meshed audio connections with which each participant establishes audio connections to all the other participants [19]. With the latter, this module must mix the received audio signals on the end host (see the "*Mixer*" module in Figure 4). Although the *Media Delivery* module (user-session dependent) under the Session Manager specifies whether mixing is turned on, based on whether a mixing server is available (from the *Call Processing* module), the actual mixing is conducted by the *Media Processing and Transmission* module.

(d) Signaling**:** The Signaling module carries out the *basic* signaling operations according to the signaling protocols, such as registration, invite/disconnect a user, media type and parameter negotiation. In contrast to the *Call Processing* sub-module under the *Session Manager*, the Signaling module is unaware of the states of a particular NCB user session, such as the mapping between a user session and SIP signaling sessions. On the other hand, the Signaling module encapsulates the signaling heterogeneity, such as different signaling protocols (SIP [9] vs. H.323 [7]), the availability of NAT traversal [18].

(e) QoS and Self Management**:** As illustrated in Figure 5, the QoS and Self-Management module autonomously monitors and adapts the behavior of the *Media Delivery* module using some embedded software sensors (for monitoring) and effectors (for adaptation), respectively. It can automatically adapt the transmission parameters and modes by seamlessly handling network transitioning and by hiding network faults. The self-management behavior of this module follows the high-level policies specified through the `applyPolicy` as the guideline from upper-layer applications (see Table 4). For example, if the available bandwidth is low, depending user/application preferences specified through high-level policies, this module can either (i) instruct the *Media Delivery* module to use encoding schemes that provide less resolution and consume less bandwidth; or (ii) suspend video transmission in order to maintain high-quality voice communication; or (iii) slow down (by decreasing socket buffer sizes) file transfer for high-quality video/audio. Some experiments showing the benefits of using this self-managing behavior are presented in Section 4.

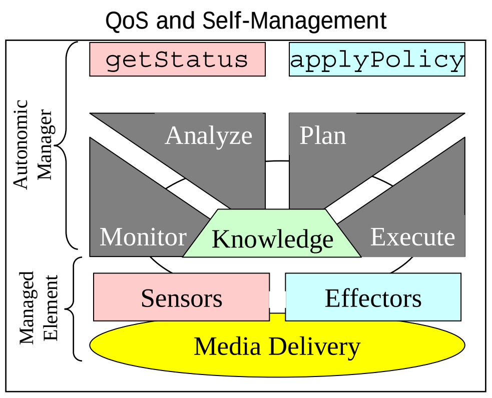


Figure 5: QoS and Self-Management internal architecture.

(f) Presence: A user *X* may need to know whether his/her friend *Y* is online in the system, indicated by login of *Y* at his/her signaling server. The user *X* may rely on mechanisms such as SUBSCRIBE/NOTIFY in SIP [9] to request the registration server to "push" the information to the NCB. Since this information does not belong to any established session, a separate module, *Presence*, is introduced for this purpose.

3.3 Discussion

3.3.1 Design Space

One design target of NCB internal architecture is to provide an extensible framework for new features. As outlined in section 3.1, separating a user-level session from the underlying network-level sessions is an important design choice, which is indicated by the horizontal dotted line in Figure 3. Only the modules above the dotted line are aware of user sessions, while all the modules below that line are responsible only for individual network-level sessions. Adding new features related to user sessions, such as "`getLastMissedCall`", will only change the *NCB Manager* and the *Session Manager* above the dotted line in Figure 3. On the other hand, adding / changing the underlying signaling protocols (e.g. changing from H.323 signaling protocol to SIP, adding NAT traversal, with / without a SIP/Presence server etc.) will only affect the *Signaling* module below the dotted line. This design principle can accommodate heterogeneous networks, and make NCB and its supported multimedia collaborative applications more portable.

While the control plane (signaling etc.) of NCB is extensible, its data plane can also accommodate networking heterogeneity. By systematically organizing and classifying media-related operations into transmission / receiving, pre- / post- processing, media I/O, and self-management (as shown in Figure 4), NCB can smoothly manage different cases (with / without conferencing mixer; real-time vs. best-effort delivery, different user preferences under various bandwidth availability), as discussed in section 3.2

We are aware of the importance of other issues, such as security, energy consumption, and mobility support. For example, each NCB session may have different security policies. However, the main focus of this paper is to demonstrate an extensible framework that facilitates hiding the communication complexity and heterogeneity, rather than new communication functionality. We do not envision any roadblocks to incorporating such advanced features with an extensible framework established.

3.3.2 Other Devolvement Factors

There are other development factors that need to be considered. These factors include (a) Licensing costs. If the use of a middleware requires expensive license, it may be more cost-efficient to implement the application from scratch. (b) Expected support for the middleware in the future, to deal with possible changes of the protocols and underlying networking infrastructure, or to add new functions or protocols. (c) Availability of open-source applications that can be tailored for free or for inexpensive license payment. Instead of using a middleware, a developer can modify an existing open-source application and integrate it in his/her software.

However, as discussed above, the main focus of this paper is to prove the concept of hiding the communication complexity and heterogeneity from high-level design collaborative applications. What we have proposed is an extensible framework, from the academic point of view. We envision that this abstraction and framework lays a foundation for the development of an open middleware. With an open middleware based on such a framework, it will require zero or low licensing cost. Meanwhile, open-source developers can add new protocols or new networking functions into the middleware without bearing in mind high-level application logic. Furthermore, it will significantly facilitate the development of open-source collaborative applications, as they can be easily ported to new networking environments. Without the separation, incorporating support of a new protocol into various collaborative applications one by one will be a tremendous effort.

4. Prototype Implementation and Evaluation

In order to evaluate the concept of NCB, we have developed a prototype of NCB in Java, called NCB/J. As the SIP protocol is accepted as a standard protocol for Voice over IP, we chose SIP as our signaling protocol. Among the implementations of the SIP protocol, we chose the open source JAIN SIP [21] by NIST. Some of the features provided by the NCB can be mapped to the existing protocol standard. For example, adding a medium in the middle of a session is supported by the SIP re-invite message. Negotiating unidirectional media transfer is implemented by the “send-only” or “recv-only” attributes of Session Description Protocol (SDP) [9]. The signaling messages of NCB go through SIP Express Router, an open source SIP server (http://www.iptel.org/ser/). For real-time multimedia transmission over IP networks, RTP is used as the transport protocol. We developed our prototype based on the JMF [20], which uses RTP. In our prototype, files are transferred via TCP connections and instant messages are delivered via SIP Messages. The prototype follows the extensible framework discussed in section 3, so that it can easily incorporate new media.

Figure 6 gives a high-level message sequence chart of user-level session management, which maps the NCB architecture shown in Figure 3 into the real implementation. It demonstrates the layer interaction on both sides: the caller (Peer A) and the callee (Peer B). The sequence chart covers the session establishment/teardown use-case as well as the network failure use-case. Each activity (*e.g.* sending a message, or notifying the applications) of user-level session management is triggered by an event (*e.g.* receiving a protocol message from the remote peer, or an invocation from upper-layer collaborative applications). It enables user-lever sessions by allowing adding/removing of media/participants any time during the life cycle of a user session. As discussed at the beginning of section 3, the Session Manager relies on Signaling Manager to manage the control-plane and the Media Manager to manage the data plane.

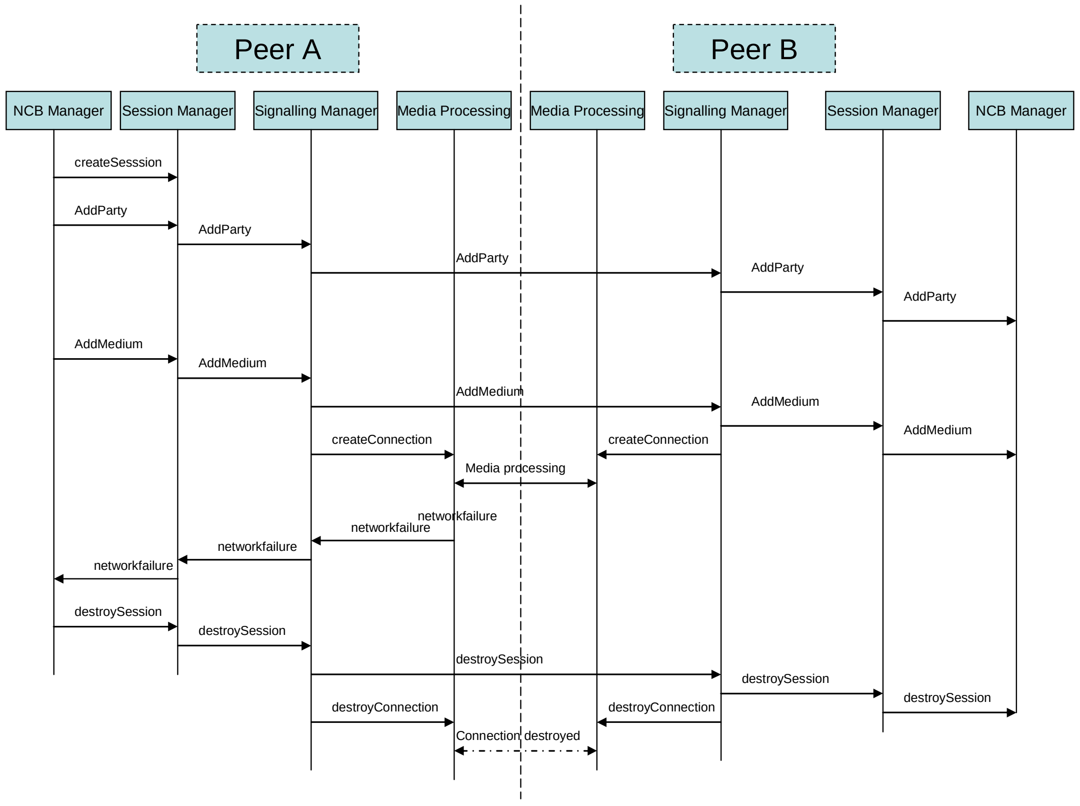


Figure 6. Message Sequence Chart of User-Level Session Management

To justify the NCB concept, we developed two types of applications based on NCB: person-to-person voice call, and person-to-person video communication (including both video and audio). We compare these against two equivalent open source applications developed upon JAIN-SIP/JMF that we downloaded off the Internet: the JAIN-SIP-Applet-Phone (https://jain-sip-applet-phone.dev.java.net/) for person–to-person voice call and the SIP-Communicator (https://sip-communicator.dev.java.net/) for person to person video communication, shown in Table 5. The encoding schemes used are G.711 and Motion JPEG, for audio and video, respectively. For each type of application, we did comparative experiments to evaluate the NCB.

Application	**Based on JAIN_SIP/JMF**	**Based on NCB**
Person-to-Person Voice Call	JAIN-SIP-Applet-Phone	NCB-based Voice Call
Person-to-Person Video Communication	SIP-Communicator	NCB-based Video Communication

Table 5: Applications and Development.

The experiments were conducted on high-end desktop computers as well as laptops. The high-end desktop computers were equipped with dual Xeon processor, 2GB memory, Gigabit Ethernet adaptors, and PCI-X bus. The Windows XP operating system was installed. The computers were connected to a campus network with high-speed routers / switches. The laptops are connected to 802.11 wireless LANs. Experiments were connected on both wired / wireless networks. To examine the self-management feature, we also tested the application behaviors with a changing available-bandwidth. For these experiments, since it's hard to control the background traffic, we conducted these experiments in dedicated VLANs and use software to limit the available bandwidth (see section 4.3 for details).

4.1 Value of High-level NCB Abstraction

We used the lines of code (*LOC*) metric to compare the above applications, with and without the NCB abstraction. The results are shown in Table 6. The development time for Person to Person Voice Call application based on NCB is about 5 hours (one developer). The development time for Person to Person Video Communication based on NCB is about 6 hours (one developer). We did not get the development time for the open source applications. However, based on the lines of code comparison with and without NCB, it is reasonable for us to conclude that the development time for communication applications without NCB would be significantly longer, probably requiring several days. The experiments show that in terms of the lines of code (*LOC*) metric and the development cycle, the NCB interface makes it significantly easier to develop multimedia communication applications.

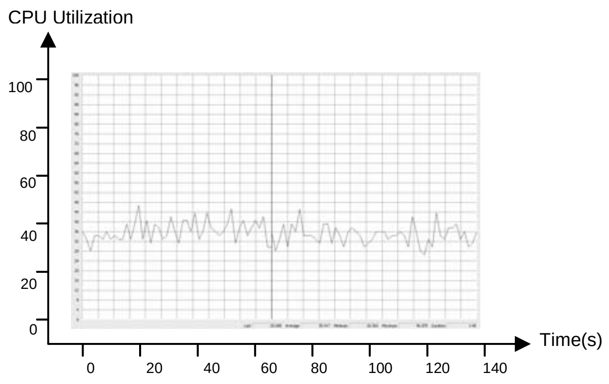
Application	JAINSIP/JMF (LOC)	NCB (LOC)
Person to Person Voice call	9478	435
Person to Person Video Communication	16784	440

Table 6: Lines of Code comparison for developing applications with/without the NCB abstraction**.**

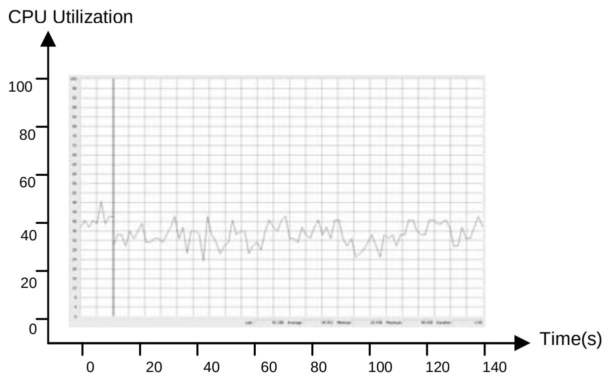
4.2 Performance Evaluation

While providing a higher-level abstraction to collaborative applications, NCB could potentially introduce performance overhead. Although NCB does not touch the network protocols and infrastructure, it changes the paradigm of application development on end hosts. Therefore, it is very important to evaluate the NCB performance on end-hosts. We compare the CPU utilization as well as the network utilization of the applications developed with and without NCB. We conducted the experiments with both high-end desktops connected to high-speed networks, as well as laptops connected to wireless LANs with relatively low bandwidth. Both results demonstrate that NCB can provide the higher-level abstraction without compromising the performance.

For person to person voice call, the average CPU utilization is around 0.237% with the JAIN_SIP-Applet-Phone application, and 0.284% with the NCB-based equivalent application. Figure 7 shows the CPU utilization over time collected with Windows Performance (the vertical red line is just an artifact of the software). For person to person video communication, the average CPU utilization is 35.417% with SIP-Communicator, and 34.912% with the NCB-based equivalent application. The CPU utilizations are almost the same.



(a) SIP-Communicator



(b) NCB-based software.

Figure 7. CPU Utilization with Person-to-Person Video Communication

In terms of network utilization, for person to person voice call, the average throughput is 73.8 kbps with the JAIN-SIP-Applet-Phone, and 73.2 kbps with the NCB-based equivalent application. For person to person video communication, the average network bandwidth of SIP-Communicator is 830 kbps, and 670 kbps for the NCB-based application, due to an optimized image compression rate of NCB discussed in section 4.3.

4.3 Self-Management Experiments

Next, we demonstrate how NCB supports self-optimization as one aspect of self-management. The high-level policy from the upper-layer application reflects the user preferences: *if the network bandwidth changes, then adapt the video compression rate and hence the image resolution.* This policy implies that the frame rate should be stable (in this case 13 fps). This high-level policy is expressed using the XML policy string as shown in Figure 2. The throughput of video traffic, limited by the available bandwidth, is the product of the frame-rate and the frame-size (which affects the image resolution). The latter is further determined by the image compression rate. Without the self-managing policy at the sender, a decreased bandwidth will cause packet losses, and significantly reduce the frame-rate at the receiver side. With the above policy, the user can expresses his/her preference to maintain the stable frame-rate at the expense of the frame-size and the image resolution: if the available bandwidth decreases, increase the image compression rate to reduce the frame-size; if the available bandwidth increases, decrease the compression rate to increase the image resolution. The compression rate is controlled by setting quantization parameters during the encoding process.

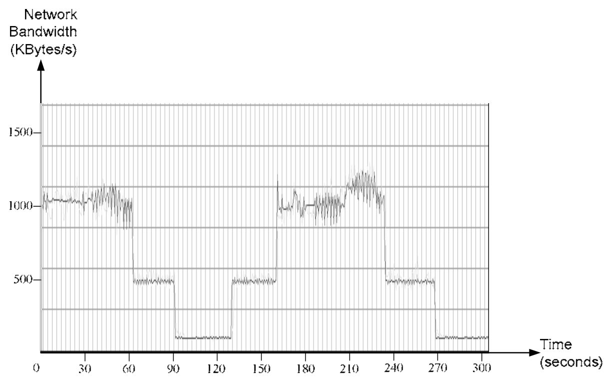


Figure 8: The network bandwidth (deep blue line) and video stream (light blue line) over time.

To simulate the change of bandwidth, we use NetPeeker (http://www.net-peeker.com/), a network speed limiter, to control the traffic. With NetPeeker, we simulate three network capacities: 1100KB/s, 500KB/s, and 100KB/s. The results of this experiment are illustrated in Figure 8. The deep blue line shows the network bandwidth and the light blue line represents the sending rates of video stream. As Figure 8 shows, NCB dynamically adjusts its video sending rates based on the change of available bandwidth.

Finally, we performed the same experiment with the SIP-Communicator and compared its receiver-side frame-rate with an equivalent NCB-based application, shown in Figure 9. The red O symbols represent the frame rates for the SIP-Communicator at different network bandwidths while the blue X symbols represent the frame rates of NCB-based implementation. With the configured policy, the frame rate of NCB is stable when the network bandwidth decreases, due to the increased compression rate. With a fixed compression rate, the receiving frame rate of SIP- Communicator decreases and sometimes the video freezes, due to packet losses. Without the policy, the behavior of NCB is the same as SIP-Communicator.

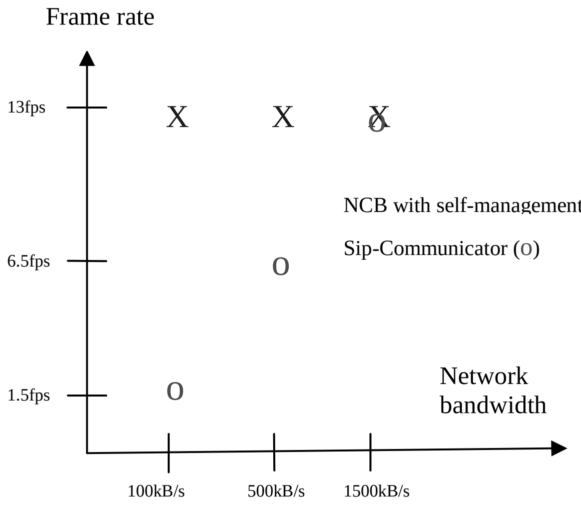


Figure 9: Frame rate changes with network bandwidth change.

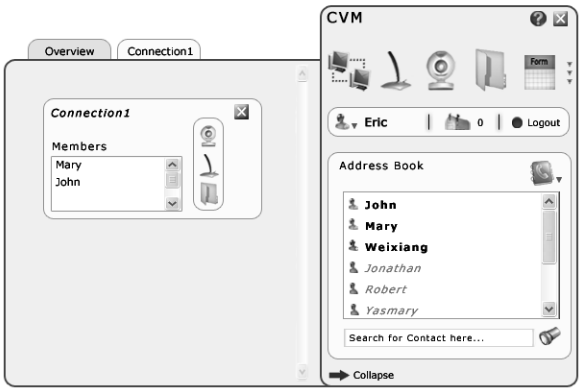
Note that it's not that self-management cannot be built into the non-NCB applications developed with a stovepipe approach. Rather, based on NCB, no-matter the application logic, developers can simply pass the policy shown in Figure 2 to achieve the self-management without changing the underlying code. That is, NCB makes the feature of self-management more reusable.

4.4 NCB as a Layer in CVM

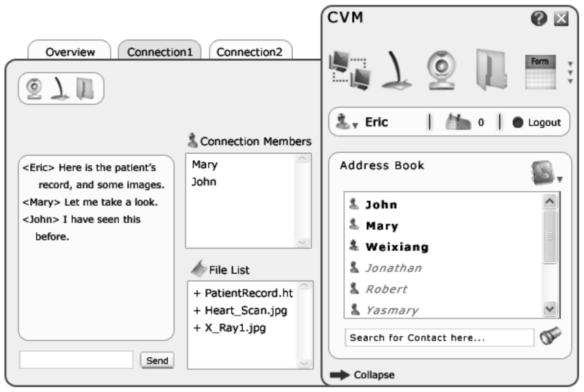
As mentioned in the introduction, NCB realizes the lowest layer of abstraction inside the CVM layered architecture [6]. To evaluate the effectiveness of NCB, we have incorporated a Java-based implementation of NCB (NCB/J) in the CVM prototype.

Figure 10 shows two screenshots of CVM prototype that illustrate how easily a Telemedicine application can be made readily available through the CVM generic Web-based GUI and model-driven schema. This figure captures a scenario within which Mary loads the Telemedicine communication schema (which contains the application-dependent collaborative logic) from the schema repository, and selects the two participants (Eric and John) from her *Address Book* (Figure 10 (a)). The media used in the connection are selected from the Media Library (represented by icons on the top right of Figure 10 (b)), and the two JPG files ("Heart_Scan.jpg" and "X_Ray1.jpg"), are dragged into the Connection Box by Eric during the conversation (Figure 10 (b)). The actual communication delivery is performed by the underlying application-independent NCB/J.

The file sharing service of this Telemedicine application uses the SCP protocol, while the Web GUI uses the HTTP protocol (see Figure 3), in contrast to the SIP/RTP protocol used in voice communication. The extensible media processing framework discussed in Figure 4 internally translates different medium requests into different pre-/postprocessing and transmission operations, based on the underlying protocols. Hence, NCB can integrate both real-time communication and best-effort delivery under the unified abstraction of user-level session presented in section 2.



(a) Overview of active communications.



<Eric> Here is the patient's record, and some images.
<Mary> Let me take a look.
<John> I have seen this before.

(b) Details of a particular active connection.

Figure 10. Screenshots of CVM prototype.

5. Related Work

Prior work related to NCB can be categorized into three major groups. In the rest of this section, we briefly introduce projects in each category and discuss how they are related to NCB.

Multimedia Communication Applications. MSN Messenger, Jabber [23], Google Talk [24] and Skype [25] are among the numerous multimedia communication applications that are being widely used. These applications provide a one-size-fits-all solution to multimedia communication and fail when there is a need for more specialized communication. Such generic multimedia applications can be developed rapidly using NCB without worrying about the complexity of network-level programming, as shown in section 4.4.

There are other projects including Polycom [26], RADVision [27], and Access Grid [28] that propose various IP-based conferencing services. We consider these approaches as complementary to NCB and we plan to benefit from some of their services. For example, we plan to use the VRVS reflectors to provide scalability to NCB.

In addition, most of these applications are non-open applications with license requests. We envision that the open architecture NCB can promote the development of an open-source and portable middleware. Therefore new communication features can be added into this customizable middleware without knowing the future application logic.

Protocols, APIs, and Software Frameworks. TCP/UDP socket is a low-level abstraction that we have already compared in subsections 2.1 and 3.1. SIP [9] and H.323 [7] are among the signaling protocols for Internet telephony, while RTP [15] provides transport functions for transmitting real-time audio and video. Java Media Framework (JMF) [20] is a library for audio and video delivery. JAIN SIP [21] is a standardized Java interface to SIP. The low-level APIs of these communication libraries are still significantly complex to use, and do not encapsulate the concept of user-level sessions (discussed in section 2.1) for multimedia multi-party communication. In fact, NCB internally uses these protocols and frameworks to implement its high-level abstractions. Furthermore, these low-level APIs do not hide the complexity and the heterogeneity of underlying networks.

The Java Telephony API [29] is a high-level API for traditional telephony applications, and does not support *multimedia* communication applications with sophisticated communication needs. [2] discusses open software architectures for IP-based voice communication. Parlay [22] is an interface that enables the rapid creation of telecommunication services. These frameworks ([2, 22]) mostly address server-side architectures. The server-side architecture has different concerns than the client-side middleware, which is the focus of NCB. Furthermore, in contrast to traditional telephone networks, where end devices are "dumb", in IP networks, end-hosts are capable of sophisticated collaborative logic. Therefore, client-side middle development is as important and complicated.

There are a number of other communication middleware proposed recently [30, 31, 32, 33]. [30] focuses on the object-oriented design of middleware purely from the perspective of software engineering, without considering issues in multimedia multi-party communication. While [31]'s scope is disaster management in mobile networks, [32]'s scope is on pipelined communication in cluster computing. [33] improves the efficiency of group communication. None of theses works have a general scope on the middleware development for the next-generation multimedia telecommunication in collaborative applications.

Reflective and adaptive middleware and toolkits. In order to provide self-management in software, two general approaches have been used: parameter and compositional adaptation [13]. *Parameter adaptation* involves the modification of variables that determine program behavior [5, 8, 11, 17]. A weakness of parameter adaptation is that it cannot adopt algorithms or components left unimplemented during the original design and construction of an application. That is, parameters can be tuned or an application can be directed to use a different existing strategy, but strategies implemented after the construction of the application cannot be adopted. In contrast, *compositional adaptation* results in the exchange of algorithmic or structural parts of the system with ones that improve a program's fit to its current environment [1, 4, 12, 14].

In its internal design, NCB employs both parameterized and compositional adaptation. Instead of reinventing the wheel, NCB incorporates existing adaptive and reflective middleware toolkits to provide self-management using only high-level policies reflecting user or application preferences. ACE, Ensemble, and Open ORB are among the projects that we closely follow to incorporate some of their services inside NCB. ACE [16] is a real-time object-oriented framework written in C++ that wraps many OS services and provides a variety of communication-related patterns that can be employed by NCB. Ensemble [14] is a groupware communication toolkit that supports protocol stacks constructed from fine-grained components, called micro-protocols. We plan to leverage these micro-protocols to provide the desired behavior, with respect to the high-level policies, for NCB at runtime. OpenORB [3] focuses on the role of computational reflection in middleware for mobile multimedia applications that can be dynamically adapted in response to the environmental changes. NCB is also a reflective middleware, but does not focus only on mobile computing.

6. Conclusion and Future work

We have proposed NCB, a unified high-level abstraction that separates the complexities of network-level communication control and media delivery from the application-dependent collaborative logic. NCB facilitates rapid creation of portable collaborative multimedia applications. We have identified the requirements of the NCB abstraction required for the class of multi-party and multimedia communication applications. The design of NCB is based on an extensible and self-managing software framework. In the future, we plan to enhance the extensibility, reusability, and self-management of NCB.

References

1. M. Aksit and Z. Choukair, “Dynamic, adaptive and reconfigurable systems overview and prospective vision,” in Proceedings of the 23rd International Conference on Distributed Computing SystemsWorkshops (ICDCSW’03), May 2003.
2. G. W. Bond, E. Cheung, K. Hal Purdy, P. Zave, and J. C. Ramming, “An open architecture for next-generation telecommunication services”, ACM Transactions on Internet Technology IV(1):83-123, February 2004.
3. G. S. Blair, G. Coulson, P. Robin, and M. Papathomas, “An architecture for next generation middleware”, Middleware’98, September 1998.
4. W. K. Chen, M. A. Hiltunen, and R. D. Schlichting, “Constructing adaptive software in distributed systems,” in Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS-21), pp. 635–643, April 2001.
5. A. K. Dey and G. D. Abowd, “The context toolkit: Aiding the development of context-aware applications,” the 22nd International Conference on Software Engineering (ICSE), June 2000.
6. Y. Deng, S. M. Sadjadi, P. Clarke, C. Zhang, V. Hristidis, R. Rangaswami, and N. Prabakar. A communication virtual machine. the 30th Annual International Computer Software and Applications Conference (IEEE COMPSAC), September 2006.
7. ITU-T Recommendation H.323v.4 "Packet-based multimedia communications systems", November 2000.
8. M. A. Hiltunen and R. D. Schlichting, “Adaptive distributed and fault-tolerant systems,” International Journal of Computer Systems Science and Engineering, vol. 11, pp. 125–133, September 1996.
9. M. Handley, H. Schulzrinne, E. Schooler and J. Rosenberg, “SIP: Session Initiation Protocol”, RFC 2543, March 1999.
10. J. O. Kephart and D. M. Chess, “The vision of autonomic computing,” IEEE Computer, vol. 36, pp. 41–50, January 2003.
11. G. Kortuem et. al, “When peer-to-peer comes face-to-face: Collaborative peer-to-peer computing in mobile ad-hoc networks,” in Proceedings of the 2001 International Conference on Peer-to-Peer Computing (P2P2001), August 2001.
12. P. K. McKinley, U. I. Padmanabhan, N. Ancha, and S. M. Sadjadi, “Composable proxy services to support collaboration on the mobile internet,” IEEE Transactions on Computers, pp. 713–726, June 2003.
13. P. K. McKinley, M. Sadjadi, E. P. Kasten, and B. H. C. Cheng, “Composing adaptive software”, IEEE Computer, pages 56-64, July 2004.
14. R. van Renesse, K. P. Birman, M. Hayden, A. Vaysburd, and D. Karr, “Building adaptive systems using Ensemble,” Software Practice and Experience, vol. 28, August 1998.
15. H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, “RTP: A Transport Protocol for Real-Time Applications”, RFC 3550, July 2003.
16. D. C. Schmidt, “The ADAPTIVE Communication Environment: An object-oriented network programming toolkit for developing communication software,” Concurrency: Practice and Experience, vol. 5, no. 4, pp. 269–286, 1993.
17. J. P. Sousa and D. Garlan, “Aura: an architectural framework for user mobility in ubiquitous computing environments,” in Proceedings of the third Working IEEE/IFIP Conference on Software Architecture, pp. 29–43, 2002.
18. J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy, “STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)”, RFC 3489, March 2003.

19. Jonathan Lennox and Henning Schulzrinne, “A Protocol for Reliable Decentralized Conferencing”, ACM NOSSDAV 2003.

20. Java Media Framework (JMF). http://java.sun.com/javase/technologies/desktop/media/jmf/

21. JAIN-SIP. https://jain-sip.dev.java.net/

22. The Parlay Group, Parlay/osa specifications, http://www.parlay.org/en/specifications/.

23. Jabber, http://www.jabber.org/.

24. Google, Google Talk. http://www.google.com/talk/.

25. Skype Limited, Skype developer zone. https://developer.skype.com/.

26. Polycom, http://www.polycom.com/

27. Radvision, http://www.radvision.com/

28. Access Grid, http://www.accessgrid.org/

29. SUN, Java Telephony API, http://java.sun.com/products/jtapi/

30. F. M. Q. Pereira1, M. T. O. Valente, R. S. Bigonha1 and M. A. S. Bigonha1, “Arcademis: a Framework for Object Oriented Communication Middleware Development”, Software: Practice and Experience. Volume 36. Issue 5. pp. 495-512. 2006.

31. L. Juszczyk, S. Dustdar, “A Middleware for Service-oriented Communication in Mobile Disaster Response Environments”, 6th International Workshop on Middleware for Pervasive and Ad-Hoc Computing (MPAC). 9th ACM/IFIP/USENIX Middleware Conference 2008. December 2008.

32. S. Fide and S. F. Jenks, “A middleware approach for pipelining communications in clusters,” Cluster Computing, vol. 10, pp. 409–424, December 2007.

33. Jianjun Zhang, Ling Liu, Lakshmish Ramaswamy, Gong Zhang and Calton Pu, “A Utility-Aware Middleware Architecture for Decentralized Group Communication Applications”, ACM/IFIP/USENIX 8th International Middleware Conference (MIDDLEWARE-2007), November 2007

34. C. Zhang, M. Sadjadi, W. Sun, R. Rangaswami and Y. Deng, "A User-Centric Network Communication Broker for Multimedia Collaborative Computing", Proceedings of the 2nd International Conference on Collaborative Computing, Atlanta, November 2006.