A User-Centric Network Communication Broker for Multimedia Collaborative Computing

Chi Zhang, S. Masoud Sadjadi, Weixiang Sun, Raju Rangaswami, Yi Deng
School of Computing and Information Sciences
Florida International University
{czhang, sadjadi, wsun001, raju, deng}@cis.fiu.edu

Abstract—The development of collaborative multimedia applications today follows a vertical development approach, which is a major inhibitor that drives up the cost of development and slows down the pace of innovation of new generations of collaborative applications. In this paper, we propose a network communication broker (NCB) that provides a unified higher-level abstraction that encapsulates the complexity of network-level communication control and media delivery for the class of multimedia collaborative applications. NCB expedites the development of next-generation applications with diverse communication logics. Furthermore, NCB-based applications can be easily ported to new network environments. In addition, the self-managing design of NCB supports dynamic adaptation in response to changes in network conditions and user requirements.

I. 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 communication applications. However, the fast pace growth of innovations has been restrained by the stovepipe approach currently employed in application development. 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 are primitive and often heterogeneous. Further, the underlying network configurations can also vary significantly which can reduce application portability.

What is lacking is a shared and systematic approach to design across various collaborative applications. In [6], we introduced Communication Virtual Machine (CVM) that represents a paradigm shift in how a collaborative application is conceived and delivered. Through its layered architecture and model-driven engineering, CVM supports 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.

In this paper, we focus on the basic application-independent communication service. We propose Network Communication Broker (NCB), a client-side middleware that encapsulates the networking complexity and heterogeneity of basic multimedia and multi-party communication for diverse upper-layer collaborative applications, ranging from a simple phone call and video conferencing to specialized communication applications like scientific collaboration, disaster management, and telemedicine. The key innovation of the NCB concept is a unified communication abstraction that separates and isolates the complexities of network-level communication control and media delivery from the diversity of application-dependent collaboration logic. Under this unified high-level abstraction, internally NCB translates a high-level communication task into a series of operations, and coordinates the underlying heterogeneous network infrastructure, systems and libraries (such as TCP/UDP sockets, SIP-based signaling, RTP-based real-time media transport, and device drivers) to smoothly carry out communication tasks. The scope of NCB abstraction is limited to providing a user-centric multi-party and multimedia communication service.

The values of this unified NCB abstraction are (a) the unified abstraction provides generic user-centric communication services reusable by a wide variety of collaborative applications. (b) Since the NCB abstraction encapsulates networking complexities, the high-level application-dependent collaboration logic becomes relatively simple to build. (c) NCB hides the 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 abstraction and provide an API that exemplifies this abstraction. In terms of approaches, the paper has made two novel contributions. We identify that the concept of *user-level sessions* (vs. network-level sessions adopted by the existing protocols) is critical to a flexible and reusable NCB design to support next-generation collaborative communications and accommodate heterogeneous networks. Furthermore, we demonstrate how the self-managing design of NCB supports dynamic adaptation in response to changes in network conditions and user requirements.

The rest of this paper is organized as follows. In section II, we identify the set of requirements for the NCB abstraction and present a minimal API. In section III, we overview NCB's internal architecture and design. Section IV introduces the prototype implementation of NCB in Java, as well as experiments and findings. Section V presents related work and section VI provides some concluding remarks.

II. NCB ABSTRACTION AND API

Finding the appropriate level of NCB abstraction 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. We now elaborate on the 4 key aspects that are fundamental to satisfy the needs of next generation collaborative applications.

Initiation and Presence Interface: The NCB abstraction must provide a registration mechanism for all users that allows a signaling server to locate users. Table I presents the proposed NCB initialization and presence interface.

TABLE I.   NCB INITIALIZATION AND PRESENCE INTERFACE.

Interface	Description
void launch()	Configures NCB
void shutdown()	Do cleanup for NCB
boolean login(String realm, String user, String passwd)	Login into signaling server
boolean logout()	Logout from signaling server

Session Interface: The abstraction should support the basic concept of a *user session*. We define a user session within 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 various media to all the other session participants on demand (*e.g.* 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 next-generation collaborative applications such as disaster management. In response to a disaster, an administrator may initiate several user sessions for different groups, since different groups may have different topics, media types (e.g., voice communication in one user session and text chat in another), priorities and levels of secrecy. Table II presents part of the session interface.

TABLE II.   NCB SESSION INTERFACE.

Interface	Description
int createSession(String comments)	Create new communication session, return a session ID.
boolean destroySession(int sid)	Destroy an existing session
public boolean add/remParty(int sid, ArrayList parties)	Add or remove new parties in a session
boolean add/remMedia(int sid, String media_type, String media_location)	Add or remove a new media in a session

Callback Interface: The interface must provide a mechanism to expose certain low-level events of networks (*e.g.* network outage) and sessions, to the applications, since under certain circumstance, 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. The NCB callback interface in Table III enhances NCB flexibility with different application logic.

TABLE III.   NCB CALLBACK INTERFACE.

Interface	Description
Void networkFailure(String nwFailure)	Notification of network failure
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
void mediaStatus(int sid, String media_type, String media_URI, int status)	Report the status of a media in a session

Self-Management Interface: NCB must internally conduct self-optimization to autonomously adapt media-delivery to the changing network conditions. The NCB self-management [10] interface, demonstrated by Table IV, allows upper-layer applications to customize (by specifying high-level policies as guidance) NCB adaptive behaviors under specific network conditions, based on user or application preferences,.

TABLE IV.   NCB SELF-MANAGEMENT INTERFACE.

Interface	Description
String getPolicyStatus()	Get the current policy in XML format
int applyPolicy(String xmlString)	Apply the self-management policy in xmlString

We developed an XML Schema to be able to specify high-level policies in XML documents. An example of a self-optimization policy in XML that is currently supported by the NCB is shown in Figure 1. 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 rate and vice-versa to maintain a steady frame-rate. See section III for more policies and section IV for experiments.

```
<session sessionID="24">

    <connectionConstraint
     condition="networkBandwidthDecreasing"
     action="decreaseVideoResolution" />
    <connectionConstraint
     condition="networkBandwidthIncreasing"
     action="increaseVideoResolution" />

</session>
```

Figure 1.   Example self-optimization policy specification.

III. NCB INTERNAL ARCHITECTURE AND DESIGN

As shown in Figure 2, the internal architecture of NCB 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 communication messages between different NCBs following standard networking protocols may have their own notions of low-level network sessions. To encapsulate various *network sessions* (e.g. audio, video etc.) within one *user session* (see section II) of NCB, NCB must internally maintain

the mapping from the user-level session ID to the network-level session IDs of the underlying protocols. In the rest of the paper, the term "session" is used to denote a NCB user session, unless otherwise stated. In Figure 2, 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. As we will show below, this extensible design can facilitate the integration of new communication features over heterogeneous network condition. We briefly describe and discuss each module as follows.

(a) NCB Manager: The NCB Manager is responsible for the initialization and the configuration of the entire NCB middleware. For example, it maintains the signaling server information (IP address etc.). 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.

(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 delegates 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 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 carries out the basic signaling. It maintains the states of the user session, such as the mapping between the user session and individual SIP signaling sessions

maintained by the *Signaling* module.

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 (port numbers and encoding/decoding schemes) before the actual media transmission. It then controls on the "Media Processing and Transmission" module to actually transmit the media.

(c) Media Processing and Transmission: This module carries out media transmission and receiving. In addition, media will be pre-processed (e.g. encoding) at the sender side, and will be post-processed (e.g. decoding) at the receiver side.

(d) Signaling: The Signaling module carries out the *basic* signaling operations according to the signaling protocols, such as registration, invite a participant, media type and parameter negotiation. The Signaling module encapsulates the signaling heterogeneity, such as different signaling protocols (SIP [9] vs. H.323 [7]), with or without NAT traversal.

(e) QoS and Self Management: As illustrated in Figure 3, the QoS and Self-Management module autonomously monitors and adapts the behavior of the Media Delivery module. The self-management behavior of this module follows the high-level policies specified through the applyPolicy interface (see Table IV) as the guideline from upper-layer applications. For example, if the available bandwidth is low, depending on user/application preferences specified through high-level policies, this module can instruct the Media Delivery module to either (i) use encoding schemes that provide less resolution; 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.
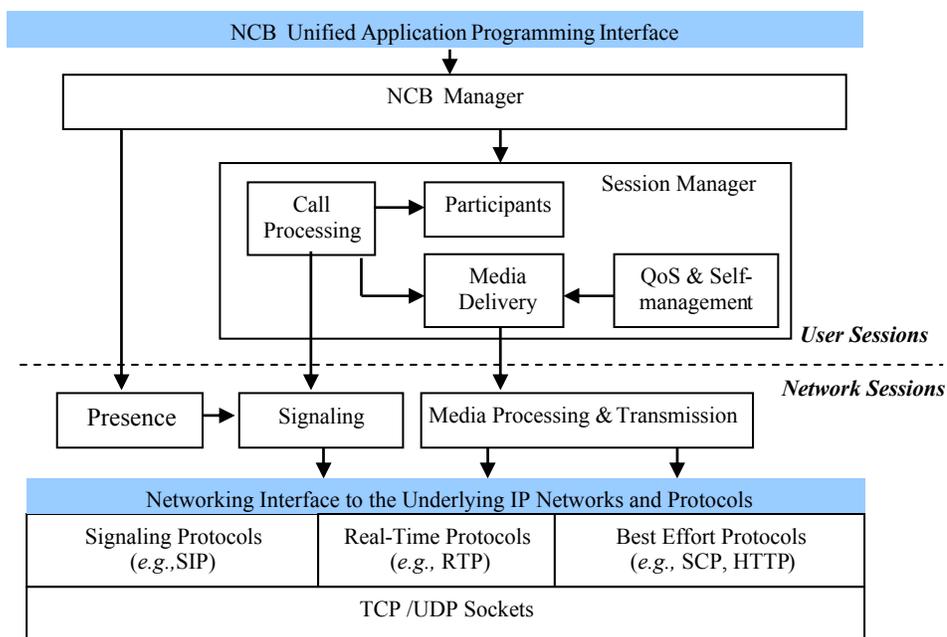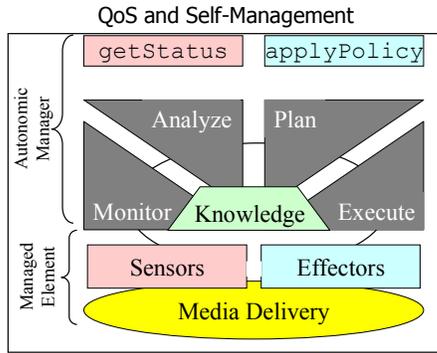

Figure 2.   The NCB architecture

Figure 3.   QoS and self-management internal architecture.

IV.   PROTOTYPE IMPLEMENTATION AND EVALUATION

We have developed a prototype of NCB in Java, called NCB/J. We chose the standard SIP as our signaling protocol, based on the open source NIST implementation of JAIN SIP. 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]. Based on JMF, RTP is used as the transport protocol for real-time multimedia. 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.

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 V. Based on the lines of code comparison with and without NCB, it is reasonable for us to conclude that the development time without NCB will be significantly longer.

TABLE V.   LINES OF CODE (LOC) COMPARISON FOR DEVELOPING APPLICATIONS WITH/WITHOUT THE NCB ABSTRACTION.

Application	Based on JAIN SIP/JMF	Based on NCB
Person to Person Voice call	9478	435
Person to Person Video Communication	16784	440

Performance Evaluation: While changing the paradigm of application development on end hosts, NCB could potentially introduce performance overhead. Our results demonstrate that NCB can provide the higher-level abstraction without increasing the CPU utilization or the network utilization. The experiment data are omitted due to the limited space.

Self-Management Experiments: Next, we demonstrate how NCB supports self-optimization that can be customized according to user preferences. The high-level policy from the upper-layer application reflects the user preferences: if the

network bandwidth changes, then adapt the video compression rate, in order to maintain a stable frame rate (in this case 13 fps). This high-level policy is expressed using the XML policy string as shown in Figure 1. Without the self-managing policy for 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 express his/her preference to maintain the stable frame-rate at the expense of the frame-size and hence the image resolution.
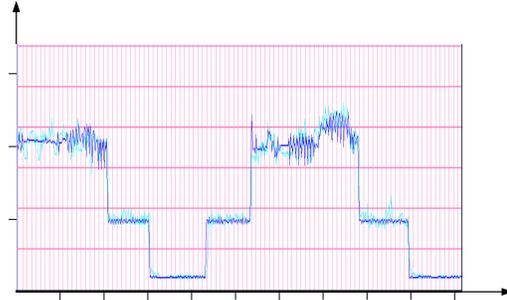

Figure 4.   The network bandwidth (deep blue line) and video stream (light blue line) over time.

We use NetPeeker (http://www.net-peeker.com/), a network speed limiter, to simulate three network capacities: 1100KB/s, 500KB/s, and 100KB/s. As shown by the results in Figure 4, NCB dynamically adjusts video throughput based on the change of available bandwidth. We also performed the same experiment with the SIP-Communicator and compared its receiver-side frame-rate with the equivalent NCB-based application, shown in Figure 5. With the configurable 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.
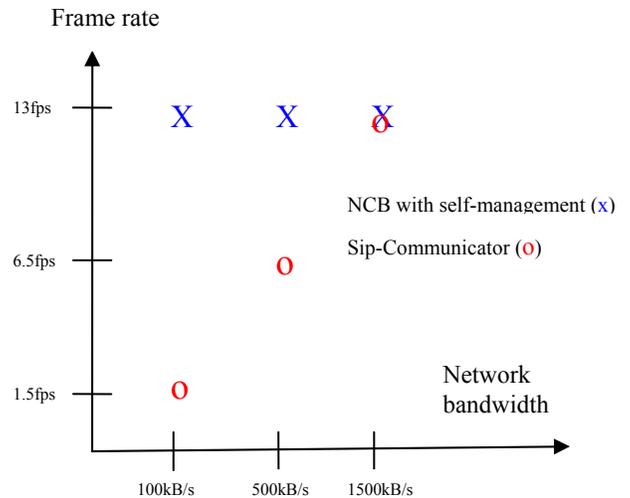

Figure 5.   Frame rate changes with network bandwidth change.

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 NCB/J into a CVM prototype. Figure 6 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 two participants (Eric and John). The media used in the connection are selected from the *Media Library* (represented by icons on the top right of Figure 6), and the two JPG files ("Heart_Scan.jpg" and "X_Ray1.jpg"), are dragged into the Connection Box by Eric. The actual communication delivery is performed by the application-independent NCB/J.

Figure 6.   Screenshots of CVM prototype: an active Connection

V.   RELATED WORK

Prior work related to NCB on communication software can be categorized into three major groups.

Multimedia Communication Applications and Services. Yahoo Messenger, MSN Messenger, AOL Instant Messenger, Jabber, Google Talk, Polycom, VRVS, and Access Grid 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 applications.

Protocols, APIs, and Software Frameworks. SIP [9], H.323 [7] and RTP [14] are among the network protocols for Internet telephony and real-time audio/video. JAIN SIP, Java Media Framework (JMF), and Java Telephony API are low-level software framework that are either still significantly complex to use, or inflexible to support next-generation collaborative applications. BoxOS [2] and Parlay are frameworks addressing server-side architectures. The server-side architecture has different concerns than our client-side middleware NCB.

Reflective and Adaptive Middleware and Toolkits. Instead of reinventing the wheel, the NCB internal design employs two general approaches to self-management: parameterized [5, 8, 11, 16] and compositional [1, 4, 12, 13] adaptation. [3, 13, 15] are among the projects that we closely follow to incorporate some of their services inside NCB.

VI.   CONCLUSION AND FUTURE WORK

We have proposed NCB, a unified high-level abstraction that separates the complexities of network-level multimedia communication from the application-dependent collaborative logic. NCB facilitates rapid creation of portable collaborative applications. 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," the 23rd International Conference on Distributed Computing Systems Workshops, 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," the 21st International Conference on Distributed Computing Systems (ICDCS), 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, 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 International Computer Software and Applications Conference, 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," the 2001 International Conference on Peer-to-Peer Computing , 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]   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.

[14]   H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", RFC 3550, July 2003.

[15]   D. Schmidt, "The ADAPTIVE Communication Environment: An object-oriented network programming toolkit for developing communication software," Concurrency: Practice and Experience, vol. 5, no. 4, 1993.

[16]   J. P. Sousa and D. Garlan, "Aura: an architectural framework for user mobility in ubiquitous computing environments," the 3rd Working IEEE/IFIP Conference on Software Architecture, pp. 29–43, 2002.