User-Centric Communication Middleware

Technical Report FIU-SCIS-2005-11-01
November 2005

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 communication 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 pace of innovation of new generation of communication applications. In this paper, we propose a *user-centric communication middleware* (UCM) that provides a unified higher-level abstraction for the class of multimedia communication applications. We investigate the minimum set of necessary requirements for this abstraction from the perspective of next-generation communication applications, and provide an API that exemplifies this abstraction. We demonstrate how UCM encapsulates the complexity of network-level communication control and media delivery. Further, we show how its extensible and self-managing design supports dynamic adaptation in response to changes in network conditions and application requirements with negligible overhead. Finally, we argue that UCM enables rapid development of *portable* communication applications, which can be easily deployed on IP-based networking infrastructure.

Keywords: Multimedia communication applications, user-centric middleware, autonomic computing.

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 communication applications. Examples include Video Conferencing, Voice over IP (VoIP), and Instant Messaging. These communication applications have the potential to dramatically impact our everyday life and change the way we communicate with each other. However, the fast pace growth of innovations in this direction has been restrained by the stovepipe approach that is currently employed in the development process of these applications. In this paper, we introduce a novel user-centric communication middleware (UCM) that provides a high-level communication abstraction; essentially, speeding up the development of new communication applications and freeing the developers from the complexity of network-level communication control and media delivery.

Today, the development of *domain-specific* communication application (*e.g.,* Telemedicine and Disaster Management) 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 communication 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 shared and systematic approach to design and development across communication applications. As a result, development continues in an ad-hoc manner creating a fragmented and non-reusable set of software products. This fundamental approach of vertical and stovepipe development, therefore, drives up the cost of development and deployment, limits the utility of the applications, and significantly slows down the pace of innovation of next generation communication applications.

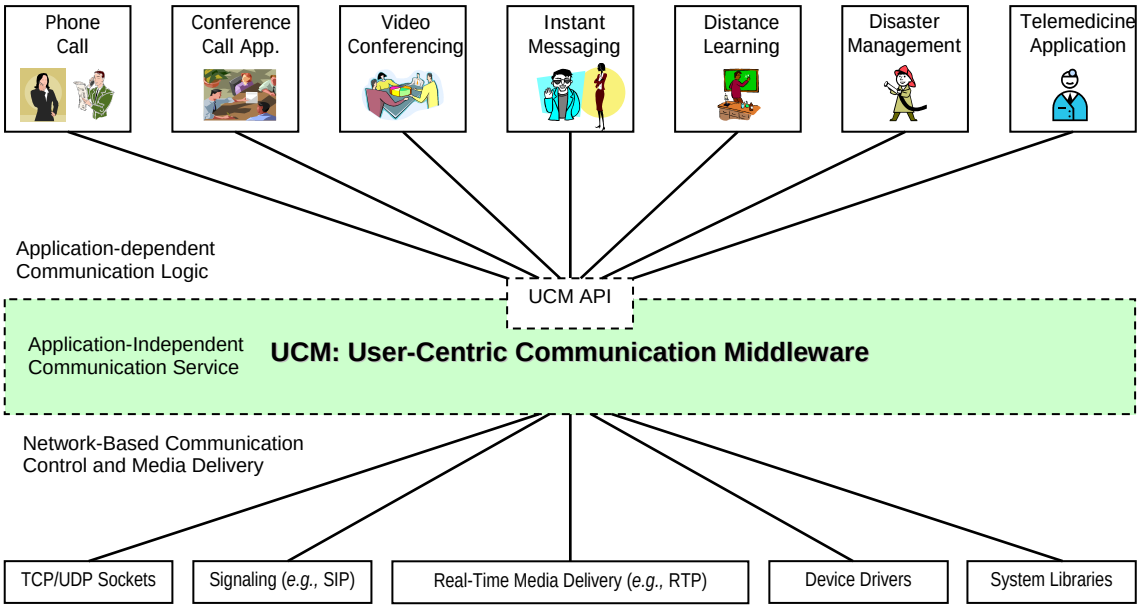

Figure 1: UCM provides an abstraction that separates network complexity from application communication logic.

Our UCM fundamentally changes the process of building communication applications from stovepipe development to horizontal middleware-based integration, which helps to streamline standardization. As illustrated in Figure 1, UCM provides a unified user-centric communication service to diverse upper-layer communication applications ranging from simple phone call and video conferencing to specialized communication applications like disaster management and telemedicine. UCM is well-positioned to serve as a client-side "middleware" that encapsulates the complexity of multimedia telecommunication. The key innovation of the UCM 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 communication logic. UCM delivers this high-level abstraction by providing a uniform and easy-to-use API that follows the Façade design pattern[1] [FHJV95]. Under this unified high-level abstraction, internally UCM utilizes the underlying network infrastructure, systems and libraries to ensure that basic communication tasks are carried out smoothly.

Since the UCM solely targets its abstraction towards providing user-centric multimedia communication service, its API has a limited scope. We identify the scope of UCM more specifically to multimedia telecommunication between users. This reduced scope allows for a simple and easy-to-use UCM interface. It also allows for policy-based self-management and autonomic behavior within the UCM. To show the value of UCM, we developed a prototype implementation of UCM in Java (UCM/J). We conducted a number of case studies using UCM/J (the results reported in later sections). Compared to previous abstractions for communication in Java, most notably the socket API in the Java network package, the signaling API of JAIN-SIP package [JAIN], and the real-time media delivery API in the JMF package [JMF], the UCM provides more coarse-grained reusability as well as ease-of-use. However, it sacrifices some flexibility due to its restricted scope of user-centric multimedia communication support for client-side applications (*e.g.*, JMF can be used to implement a media streaming server). This is illustrated in Figure 2.

In this paper, we investigate the minimum necessary requirements for the UCM abstraction, from the perspective of sophisticated communication applications, and provide an API that exemplifies this abstraction and supports dynamic multi-party and multimedia communications. We demonstrate how the extensible and self-managing design of UCM supports dynamic adaptation in response to changes in network conditions and application requirements. Further, we discuss how UCM enables rapid development of *portable* communication applications that can be easily deployed over commodity IP-based networking infrastructure. We summarize the values of this unified UCM abstraction as follows:

[1] Façade design pattern provides a simple interface to a complex system.

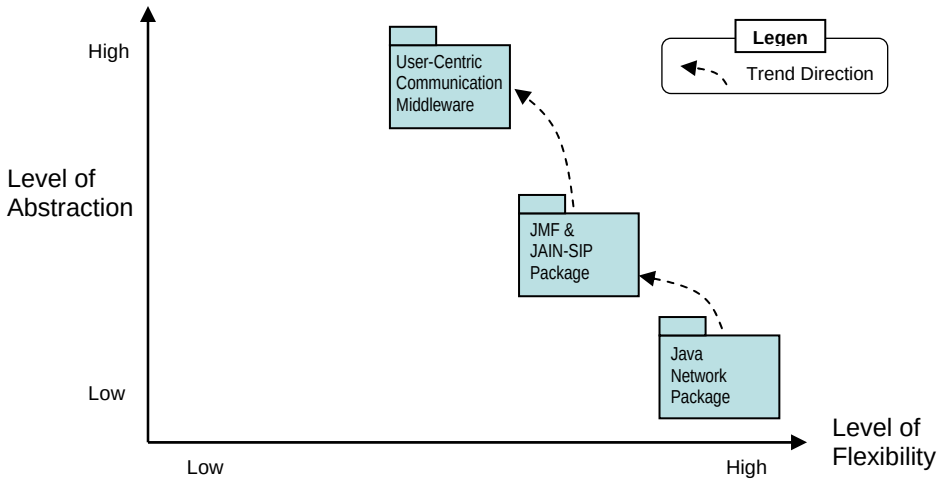

Figure 2. UCM abstraction and flexibility compared to JMF, JAIN-SIP, and Java Network Package.

- Due to the high-level abstraction separating the application-dependent business logic from the network-level basic communication services enabled by UCM, the applications are transparent to the details of underlying network protocols and infrastructure. They only deal with high-level application-dependent communication logic; which are relatively simple to express.
- Since UCM is responsible only for the basic multimedia communication services that can be shared across applications, the high-level abstraction of UCM is independent of application business logic. As a result, a version of the UCM developed and optimized for a particular system and network configuration is reusable by a variety of communication applications.
- The network infrastructure underlying the UCM can be heterogeneous, requiring different patterns of communication. For example, even for simple VoIP conferencing, the signaling procedure depends largely on the network infrastructure [LS03, Ros05]. With UCM, a change in the underlying network infrastructure does not affect the communication logic of applications; a unified UCM abstraction hides network heterogeneity from the applications so that applications can be easily ported to new network environments.

The rest of this paper is organized as follows. In Section 2, we identify the set of requirements for the UCM abstraction and present a minimal API for UCM that reflects these requirements. In Section 3, we overview UCM's internal architecture and design and show how this architecture realizes its API. In Section 4, we introduce the prototype implementation of UCM in Java and report some of our findings through the experiments we conducted using the prototype. Section 5 presents related work and Section 6 makes concluding remarks.

2 UCM Abstraction and API

The UCM abstraction is the key contribution of our work. In fact, we realized that finding the right level of abstraction is non-trivial. An abstraction that is too high-level can reduce the flexibility afforded to the application in the kind of functionality it provides to users. On the other hand, an abstraction that is too low-level, although providing a greater degree of flexibility to the application, can significantly complicate the task of the developer, increase development time, and also reduce portability. In determining the level of abstraction, we took into account the above factors as well as kept in mind the target class of applications that would use the abstraction: client-side multimedia communication applications.

In this section, first, we identify the set of necessary requirements for the UCM abstraction. We then present an API that reflects the UCM abstraction. Finally, we provide a justification as to why we believe that the proposed UCM API reflects the minimum set of necessary requirements to deliver the UCM abstraction.

2.1 UCM Abstraction Requirements

We first look at the requirements of the UCM abstraction from the perspective of upper-layer applications. 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. For example, in a simple traditional

person-to-person call, only one voice conversation needs to be supported at each end-host. With sophisticated next-generation communication applications, a user at one end-host might simultaneously communicate with different groups of people using diverse media.

To satisfy the communication needs of next generation multimedia communication applications, the UCM abstraction should support the basic concept of a *user session* and should allow for creation of multiple user sessions. We define a user session within UCM 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. In case one participant needs to communicate with only one of the other participants, a *separate* user session should be established. Within each user session, the participant should be able to deliver various media on demand, such as send a document in the middle of a voice communication. One important requirement for the UCM abstraction is supporting multiple user sessions, which is necessary for sophisticated communication 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.

Another requirement of the UCM abstraction is that it must comprehensively encapsulate the details of the end-host networking infrastructure. The abstraction should ensure that applications can be ported easily to different end-host hardware and different underlying network infrastructure. However, although the UCM encapsulates networking details, 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. Further, in some applications, the user may desire to dynamically control the behavior of UCM in session control and media delivery, according to his/her preferences, by specifying policies. The UCM abstraction must be flexible enough to support such requirements. We elaborate on these discussions further in Section 2.5.

Given the above expectations from UCM as a user-centric middleware, we identify four major categories of requirements to be reflected by its interface providing the communication service:

- First, the interface must support the basic presence functionality so that a user can login to a signaling server, and be able to retrieve and modify his/her current contact-list.
- Second, the interface must support the basic concept of communication by adequately capturing the notion of a communication user session among *multiple* participants using *diverse* media so that the set of participants and media can be dynamically controlled.
- Third, the interface must provide a mechanism to expose low-level network and system events to the applications; essentially, enabling development of context-aware applications on top of UCM.
- Fourth, the interface must provide the capability to specify high-level self-management policies to be used as guidance inside the UCM for controlling how media is exchanged and delivered.

In the rest of this section, as a proof of concept, we introduce a UCM API as four functionally separated sets of operations that reflect the four requirements of the UCM abstraction.

2.2 UCM Initialization and Presence Interface

We first present the UCM initialization and presence interface that allows an application to register a user for communication purposes with a signaling server. The signaling server information is maintained within the UCM. Table 1 presents the proposed interface.

Interface	Description
void launch();	/* Configure UCM based on a configuration file */
void shutdown();	/* Do cleanup for UCM */
boolean login(String realm, String userName, String passwd);	/* Login at the signaling server */
boolean logout();	/* Logout from the signaling server */
boolean addContact(String displayName, String identifier);	/* Add a new contact to the contact list */
boolean removeContact(String displayName, String identifier);	/* Remove an existing contact from the contact List */

Table 1: UCM Initialization and Presence Interface.

2.3 UCM Session Interface

The session interface provides an application with the ability to create multiple independent user 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 a session for a new communication */
boolean destroySession(int sid) ;	/* Destroy an existing session */
public boolean addParty(int sid, ArrayList parties);	/* Add new parties to a session */
public boolean removeParty(int sid, ArrayList parties);	/* Remove parties from a session */
boolean addMedia(int sid, String media_type, String media_location);	/* Add a new media to a session */
boolean removeMedia(int sid, String media_type, String media_location);	/* Remove an existing media from a session */
boolean suspendMedia(int sid , String media_type, String media_location, String direction);	/* Suspend the data transmission for a media */
boolean resumeMedia(int sid , String media_type, String media_location, String direction);	/* Resume the data transmission for a media */

Table 2: UCM Session Interface.

A session ID is returned by the UCM whenever a new session is created with the createSession call from the upper layer. The session ID is then used by the upper layer to uniquely identify a user session maintained within the local UCM, 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 of the media. Diverse media types are supported (*e.g.*, real-time audio/video, instant messages, and files). In addition, there is always a default medium, for continuous conversation. In traditional telephone or conferencing applications (such as those supported by Java Telephony API), the default medium is voice. However, if a user already has one session with voice as its default medium, it's not desirable to have voice as the default medium for another concurrent user session on the same device. The UCM abstraction allows the upper-layer to specify application-dependent default medium (*e.g.*, the application can establish a second session of text chat). In addition, the traditional telephone or conferencing applications assume the media delivery is always bi-directional. UCM supports both two-way (*e.g.*, voice conversation) and one-way media transfer (*e.g.*, file transfer and distance learning) with different media formats.

2.4 UCM Callback Interface

The UCM callback interface provides a mechanism by which the UCM can notify the application of specific events within the UCM. These are then used by the application in a custom fashion.

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 in a session (busy-tone, ring-tone, join, etc.) */
void media Status(int sid, String media_type, String media_URI, int status);	/* Report the status of a media in a session */

Table 3: UCM Callback Interface.

The UCM callback interface presented in Table 3 allows UCM to report status of the network, session, participants, and media. Notifying session status to the upper-layer applications enhances UCM flexibility for

different application logic. As an example, at the caller side, after the caller dials the callee, he/she should receive either a "Busy-tone" or a "Ringing-tone", indicating whether the remote device is busy/ready before the callee picks up the phone. When a "Ringing-tone" signaling message is received by the caller, the upper layer may select a presentation (*e.g.,* a flashing icon) other 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 one session. Thus, instead of encapsulating the "Ringing-tone" message and playing an audio ring on the speaker directly, the UCM simply notifies the session status to the upper-layer for application-dependent processing.

2.5 UCM Self-Management Interface

The UCM self-management interface allows application control over UCM behavior. The application can customize UCM behavior under specific network and system conditions, based on user or application preference. The interface takes as input an XML string which describes the policy for self-management. This API is presented in Table 4.

Interface	Description
int applyPolicy(String xmlString);	/* Apply the self-management policy specified by the xmlString */

Table 4: UCM Self-Management Interface.

We developed an XML Schema to be able to specify high-level policies in XML documents. At a high-level, a UCM 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 UCM is shown in Figure 3. The policy is specific to the session with ID # 24 and it dictates that when UCM detects a low network 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.

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

Figure 3. Example self-optimization policy specification.

2.6 UCM API Justification

As mentioned earlier, the UCM API is limited to the scope of user-centric multimedia communication. However, it should be flexible enough to allow diverse and sophisticated user-centric communication logic. We claim that these set of requirements are necessary for the UCM that are required to keep the API simple, yet flexible. These requirements allow the UCM API to remain simple by providing bare minimum middleware support for multimedia telecommunication among users, while at the same time, providing such applications the flexibility of establishing full-featured communication user sessions with control over the number and identity of participants and the nature and timeliness of media exchanged during the session. Further, the API also provides a mechanism for reporting session states, error conditions, exceptions, as well as specific system and network events that the application may be interested in. The API also allows for policy-driven self-management, whereby an application can tailor the behavior of UCM under special or changed environment. This minimum API can already support a variety of next-generation communication applications. We believe that with the UCM abstraction and API we have reached a higher-level of understanding about user-centric communication services.

3 UCM Internal Architecture and Design

In this section, we detail the UCM internal design that supports the unified high-level abstraction, deals with the complexity of communication, and encapsulates underlying networking heterogeneity.

3.1 UCM Design Issues

UCM is a client-side middleware to be deployed on end hosts. Below the unified UCM abstraction, the UCM 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 UCM 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) according to the requirements of communication tasks, as well as the network configurations and conditions.

The introduction of UCM will not affect or require changes to the existing protocols and network infrastructures. The communication between peer UCMs follows various protocol standards (*e.g.*, SIP [HSSR99] for signaling, and RTP [SCFJ03] for delivering real-time media), and may rely on various communications infrastructures such as signaling servers and media gateways between peer UCMs. For example, an existing signaling server can process and forward signaling messages negotiating communication parameters such as media to be transmitted, encoding/decoding schemes, TCP/UDP port numbers, and device media capabilities. The standard protocols, such as SIP, can be adopted as the signaling mechanism between UCMs, or between UCM and signaling servers. The signaling servers also accept user registration and authenticate the users. The IP address of the signaling server and its port number for SIP signaling must be configured into the UCM during its initialization. In addition, there may be a media gateway converting diverse audio/video encoding schemes from different UCMs, and mixing real-time audio/video (sometimes called Multi-Point Control Unit, or MCU) for distributed multi-party conferencing. A mixer receives individual audio streams from all session participants and combines them into a single stream (by summing audio signals), which is then sent back to all participants. Without a mixer, the UCM of each participant has to establish point-to-point audio connections (*i.e.*, a full mesh model) to all the other participants [LS03, Ros05]. These considerations reflect the design philosophy of *not re-inventing the wheel* in UCM design and implementation.

The abstraction of UCM *user session* is a high-level and simple abstraction for multimedia and multi-party communication. While the application maintains and processes application-dependent states of a session, UCM maintains and processes the application-independent low-level communication states of a session. Just like a socket number and its associated port that hide the communication details of reliable/unreliable data delivery (*e.g.*, data packetizing, packet sequencing, and acknowledgements), the UCM session ID encapsulates the complexity of multiparty, multimedia communication. To be more specific, although the session IDs are unique within each UCM, for the same user session involving multiple participants, the UCM session IDs may be different at different UCMs of different participants. The communication messages between different UCMs follow the standard networking protocol. The messages of these networking protocols may have their own notions of sessions or session IDs, and do not contain UCM session IDs. For example, a RTP session is a single media stream between two users; without a media mixer, one participant has to send his/her audio to all the other participants in duplicated streams, each of which may have different SIP session ID and RTP session ID. To encapsulate various *network sessions* in one abstracted *user session* for our user-centric middleware, UCM must internally maintain the mapping between the UCM session ID and the session IDs of the underlying protocols. In the rest of the paper, the term "session" is used to denote a UCM user session, unless otherwise stated.

The UCM internal architecture has an extensible framework facilitating the integration of new communication functionality, new media types, and new networking primitives (*e.g.*, QoS). The internal modules of UCM are designed to be extensible and reusable for different network configurations (*e.g.*, with or without conferencing mixer discussed above; with or without NAT traversal [RWHM03]; different signaling protocols such as SIP [HSSR99] vs. H.323 [H232]). The UCM 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 UCM Internal Architecture

Below the unified API, the internal architecture of UCM is outlined in Figure 4. It includes the following modules:

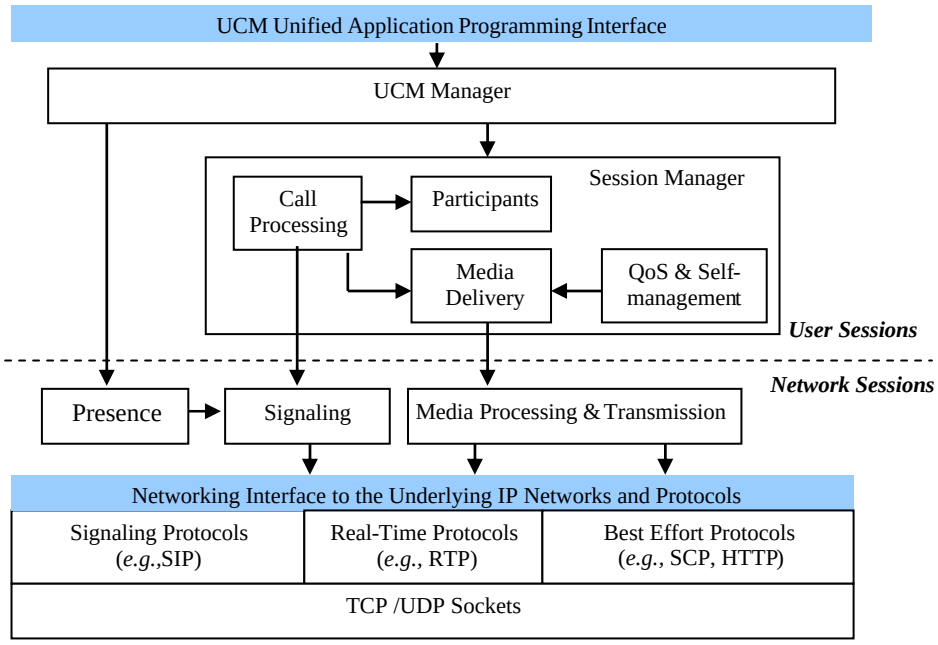

Figure 4: The UCM Architecture.

(a) *UCM Manager*:

The UCM Manager is responsible for the initialization and the configuration of the UCM middleware. The UCM configuration occurs when the UCM is initially launched and includes the signaling server information (IP address etc.). It is also responsible for registering the user account at the signaling server, providing the current 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, it creates a new Session Manager (see below) to handle the new communication session. The UCM maintains the list of Session Managers for all active sessions. In addition, the UCM manager 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 UCM 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 2.2, thus implementing the call-waiting service.

(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.

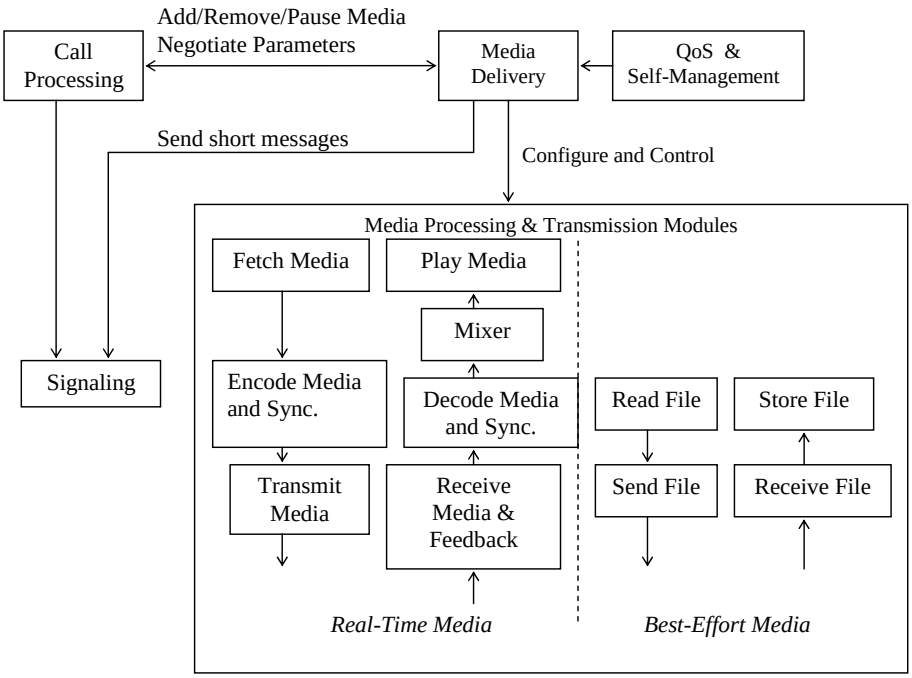

Figure 5. Signaling and Media Delivery.

The *Media Delivery* module manages, at a high level (*i.e.,* at the level of user sessions rather than network sessions), the transfer of media in a session, as demonstrated by Figure 5. It translates an "`addMedia`" call from the application into a number of internal invocations. It first relies on the Call Processing module (see below) to negotiate the transmission parameters (port number and encoding/decoding schemes) before the actual media transmission. It then controls on the "Media Processing and Transmission" module (see below) to actually deliver the media. Some media, such as short messages, can be delivered within the signaling message (*e.g.,* SIP), and thus go through the Signaling module (see Figure 5). It also triggers the delivery of all session media to a new participant, when he/she just joins the session.

The *Call Processing* module controls, at a high level (*i.e.,* at the level of user sessions rather than network sessions), the call processing logic of a session. It maintains the mapping from a user session to low-level signaling sessions. It is based on the underlying Signaling module, which actually carries out the basic signaling. For instance, it translates a call of "`addParticipant`" from the upper layer into a number of operations independent of the user session, to be delegated to the underlying Signaling module to invite the remote participant. When receiving a signaling message indicating that a new participants joins the session, it calls the Participants module to update the participant list, and then reports the newly joined participant to the upper-layer (through the `partyStatus` callback interface in Table 3). For voice conferencing, the Call Processing module adjusts its signaling procedures based on the availability of a conferencing mixer, and appropriately instructs the Media Delivery module.

(c) *Media Processing and Transmission*:

The media will be pre-processed before transmitted at the sender side, and will be post-processed and recovered at the receiver side. The processing and transmission/reception depend largely on the media types and network configurations. The *Media Processing and Transmission* module maintains the supported media types and the corresponding encoding/decoding schemes, and carries out media processing and transmission, demonstrated in Figure 5. For 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 [LS03, Ros05]. With the latter, this module must mix the received audio signals on the end host. Although the Media Delivery module (user-session dependent) under the Session Manager controls or configures whether mixing is turned on, based on the information from the Call Processing module, the actual mixing is conducted by the *Media Processing and Transmission* module. In contrast to the Media Delivery module, this module is fully unaware of the states of a user session.

As shown in Figure 5, the module has different processing paths for real-time media delivery and best-effort data delivery (*e.g.*, files). For best-effort data delivery, everything can be blindly transmitted as a file, since no encoding/decoding is needed, and transmission control is independent of the media content. At the receiver side, the received file associated with a session will be stored at a specified directory (*i.e.*, simple post-processing). Since the signaling modules has negotiated and reported the file type (*e.g.*, PDF), the upper-layer can conduct application-dependent processing, such as launching the corresponding document processing applications (*e.g.*, Acrobat Reader) according to the file type.

(d) *Signaling*:

The Signaling module carries out the *basic* signaling operations according to the signaling protocols (*e.g.*, SIP), such as registration, invite/disconnect a user, media type and parameter negotiation. In the middle of media delivery, it can also negotiate to temporarily suspend the media delivery in a session (*e.g.*, through a SIP re-INVITE message with a connection address 0.0.0.0). The major difference between the Signaling module and the Call Processing sub-module under the Session Manager is that the basic signaling activity is carried out by the Signaling module, and the Signaling module is independent of the states of a particular UCM user session. For example, the signaling module is unaware of the mapping between a user session and SIP signaling sessions. In contrast, the Call Processing module takes care of signaling issues depending on the states of the user session. On the other hand, the Signaling module encapsulates the signaling heterogeneity, such as different signaling protocols (SIP [HSSR99] vs. H.323 [H232]), with or without NAT traversal [RWHM03].

(e) *QoS and Self Management*:

This module assists Media Delivery in automatically adapting transmission parameters or modes, seamlessly handling network transitioning, and hiding or reporting network faults. The high-level policies guiding self-optimization can be given by the interface defined in Section 2.5. The policy is made concrete and implemented inside the Session Manager. For example, if the available bandwidth is low, this module can either instruct the Media Delivery module to use an encoding scheme that provides less resolution and consumes less bandwidth, or report to the upper-layer for a high-level decision (*e.g.*, instead of voice communication, use a light-weight text chat). Our experiment results are presented in Section 4.

(f) Presence

A user *X* may need to know whether his/her friend *Y* is present 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 to request the registration server to "push" the information to the client-side UCM. Since this information does not belong to any established session, a separate module, *Presence*, is introduced for this purpose.

3.3 Discussion

As can be seen, one important criterion for the UCM design is separating a user-level session from the underlying network sessions, as indicated by the horizontal dotted line in Figure 4. Each user session involves a number of network-level sessions (either signaling or media delivery sessions). Only the modules above the dotted line are aware of user sessions, while all the modules below that line are responsible for individual network-level sessions. Adding new features related to user sessions, such as "`getLastMissedCall`", will only change the UCM Manager and the Session Manager above the dotted line in Figure 4. On the other hand, changing the underlying signaling protocols (*e.g.*, from H.323 signaling protocol to SIP) will only affect the Signaling module under the dotted line.

We are aware of the importance of other issues, such as security, energy consumption, and mobility support. For example, each UCM session may have different security policies. However, the principal contribution 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 once an extensible framework is established.

4 Prototype Implementation and Evaluation

In order to evaluate the concept of UCM, we have developed a prototype of UCM in Java, called UCM/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 [JAIN] by NIST. We extended the SIP signaling protocol to negotiate transmission parameters (*i.e.*, TCP port numbers) and file names for on-demand file transfer bundled within a UCM session. However, most of the features provided by the UCM 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) [HSSR99]. The signaling messages of UCM go through SIP Express Router, an open source SIP server (http://www.iptel.org/ser/).

For real-time multimedia transmission on IP networks, RTP is used as the transport protocol. We developed our prototype based on the JMF [JMF], which uses RTP. In our prototype, files are transferred via TCP connections and instant messages are delivered via SIP Messages. The prototype follows an extensible design, so that it can easily incorporate new media.

Application	Based on JAIN_SIP/JMF	Based on UCM
Person-to-Person Voice Call	JAIN-SIP-Applet-Phone	UCM-based Voice Call
Person-to-Person Video Communication	SIP-COMMUNICATOR	UCM-based Video Communication

Table 5: Applications and Development.

To justify the UCM concept, we developed two types of applications based on UCM: 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 UCM.

4.1 High-level UCM Abstraction

By providing the high-level communication API, we claim that UCM makes it easier to develop communication applications. We used the lines of code (loc) metric to compare the above applications, with and without the UCM abstraction. The results are shown in Table 6. The development time for Person to Person Voice Call application based on UCM is about 5 hours (one developer). The development time for Person to Person Video Communication based on UCM is about 6 hours (one developer). We did not get the development times for the open source applications. However, based on the lines of code comparison with and without UCM, it is reasonable for us to conclude that the development time for communication applications without UCM 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 UCM API makes it significantly easier to develop user-centric multimedia communication applications.

Application	JAINSIP/JMF (loc)	UCM (loc)
Person to Person Voice call	9478	435
Person to Person Multimedia communication	16784	440

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

4.2 Performance Evaluation

While providing a higher-level abstraction to communication applications, UCM could potentially introduce performance overhead. Although UCM 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 UCM performance on end-hosts. We compare the CPU utilization as well as the network utilization of the applications developed with

and without UCM. Our results demonstrate that UCM can provide the higher-level abstraction without significantly 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 UCM-based equivalent application. For person to person video communication, the average CPU utilization is 35.417% with SIP-Communicator, and 34.912% with the UCM-based equivalent application, as shown in Figure 6. The CPU utilizations are almost the same; that is, the performance overhead in terms of CPU utilization with UCM-based communication applications is negligible.

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 UCM-based equivalent application. For person to person video communication, the average network bandwidth of SIP-Communicator is 830 kbps, and 670 kbps for the UCM-based application, due to an optimized image compression rate.

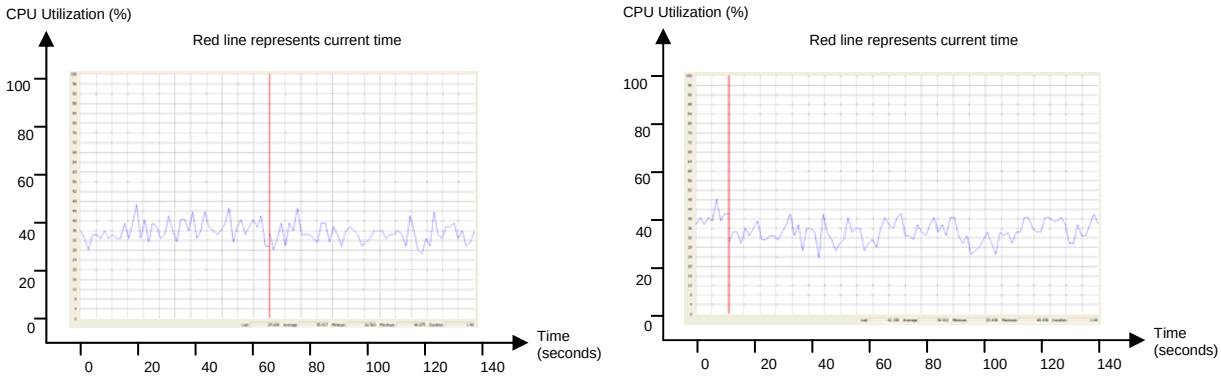

Figure 6. CPU Utilization with Person-to-Person Video Communication: (a) SIP-COMMUNICATOR (b) UCM-based software.

4.3 Self-Management Experiments

Next, we demonstrate how UCM 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 only modify the video 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 3. The throughput of video traffic, limited by the available bandwidth, is the product of the frame-rate and the frame-size. The latter is further determined by the image compression rate. 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 expresses his/her preference to maintain high frame-rate at the expense of more image compression: if the available bandwidth decreases, reduce frame resolution by increasing the image compression in order to make the frame rate stable; if the available bandwidth increases, increase frame resolution by reducing the image compression for full utilization of the network resource. Notably, before the image compression rate increases to a certain point, the difference in image quality is not discernable to many users. Second, frequent change in the compression rate may cause instability. The implementation details about setting difference thresholds are omitted due to space limitations.

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 7. The deep blue line shows the network bandwidth and the light blue line represents the throughput of video stream. As Figure 7 shows, UCM dynamically adjusts its throughput of video stream based on the change of available bandwidth.

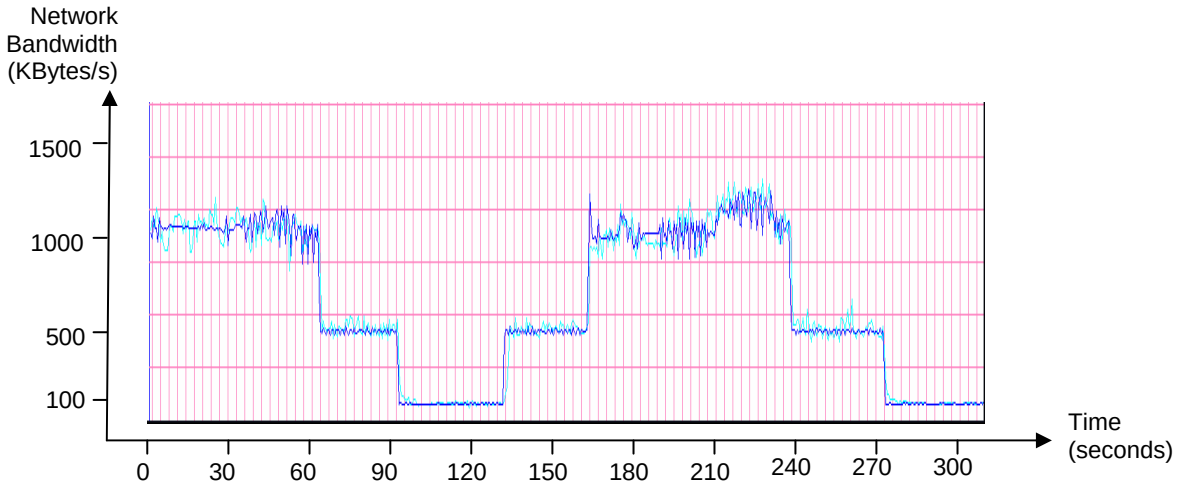

Figure 7: The network bandwidth (deep blue line) and video stream (light blue line) over time.

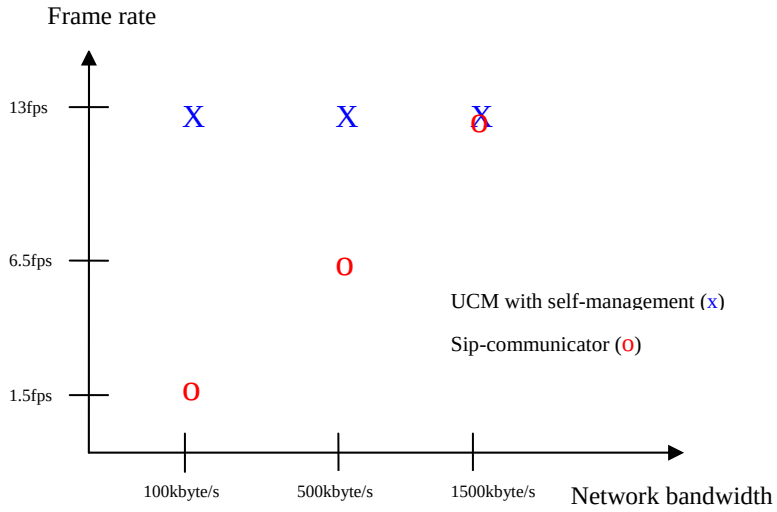

Figure 8: Frame rate changes with network bandwidth change.

Finally, we performed the same experiment with the SIP-Communicator and compared its receiver-side frame-rate with an equivalent UCM-based application, shown in Figure 8. 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 UCM-based implementation. With the configured policy, the frame rate of UCM is stable when the network bandwidth decreases, due to the increased compression rate. With a fixed compression rate, the frame rate of SIP communicator decreases and sometimes the video freezes. Without the policy, the behavior of UCM is the same as SIP Communicator.

5 Related Work

Prior work related to UCM that can be categorized into three major groups: (1) multimedia communication applications, (2) protocols, APIs, and software frameworks for developing multimedia applications, and (3) adaptive middleware and toolkits for supporting self-management in multimedia applications. In the rest of this section, we briefly introduce projects in each category and discuss how they are related to UCM.

Multimedia communication applications. Yahoo Messenger, MSN Messenger, AOL Instant Messenger, AIM, ICQ, IRC, Jabber, and Google Talk are among the numerous multimedia communication applications that are currently being used by millions of individuals and institutions around the world. These applications provide a one-size-fit-all solution to multimedia communication and fail when there is a need for more specialized communication

requirements. For example, the requirements for domain-specific communication applications such as Telemedicine, Disaster Management, Business Conferencing, Scientific Collaboration, Distance Learning, and Battlefield Coordination can not be satisfied by the generic multimedia applications. Such generic multimedia applications can be developed rapidly using UCM without worrying about the complexity of network-level programming. The developer is only required to focus on the application communication logic leaving the complexity of network programming to UCM. As a proof of concept, we have developed a prototype for Telemedicine using UCM [DSC+04].
There are other projects including Polycom, VRVS, Access Grid, and those of [LS03, GNCS04, Ros05] that propose various IP-based conferencing systems. We consider these approaches as complementary to UCM and we plan to benefit from their findings and incorporate some of their services. For example, we plan to use the VRVS reflectors to provide scalability in UCM.

Protocols, APIs, and software frameworks. SIP [HSSR99], H.323 [H323], and MEGACO [GRR00] are among the signaling protocols for internet telephony, while RTP [SCFJ03] provides transport functions for transmitting real-time audio and video.
JAIN SIP [JAIN] is a standardized Java interface to SIP. Java Media Framework [JMF] is a library for audio and video communication. The low-level APIs of these communication libraries are still significantly complex to use. For example, JAIN SIP facilitates the generation of SIP messages, and captures the SIP syntax of Transactions and Dialogues. The signaling logic is left to the application developers. The network-level session supported by JAIN SIP is far less usable than the user-centric session of UCM. JMF does not support instant messages and file transfer, and has no concept of user communication sessions. The Java Telephony API and Microsoft Telephony API are high-level APIs for traditional telephony applications. They do not support next-generation multimedia communication applications with sophisticated business logic. Eclipse Communication Framework (ECF) is a project that facilitates the creation of communications applications on the Eclipse Platform. The framework provides APIs for secure asynchronous and synchronous messaging for communication and collaboration. The ECF project does not separate the network-level communication control and information delivery from the complexity of application-oriented communication logic.
[BCP+04, JZ98, ZGS04] discuss open software architectures for IP-based voice communication. Parlay is an API that enables the rapid creation of telecommunication services. ICEBERG project [ICEB] provides core network architecture for integrated communications. These frameworks mostly address the server-side architecture and the service creations. The server-side architecture has different concerns than the client-side middleware, which is the focus of UCM. Furthermore, in contrast to traditional telephone networks, where end devices are “dumb”, in IP networks, end-hosts are capable of sophisticated communication logic. Compared to UCM, none of the above contributions have established a unified software abstraction that allows diverse and sophisticated communication logic at the client side. The UCM concept facilitates the integration of new functions and features with diverse multimedia communication, responds to the dynamic network conditions and configurations, and enables easy encapsulation of complex, heterogeneous and dynamic network systems.

Reflective and adaptive middleware and toolkits. In order to provide self-management in software, two general approaches have been used: parameter and compositional adaptation [MSKC04]. *Parameter adaptation* involves the modification of variables that determine program behavior. As described by Hiltunen and Schlichting [HS96], a well-known example of parameter adaptation is the way that the Internet’s TCP protocol adjusts its behavior by changing values that control window management and retransmission in response to apparent network congestion [KR01]. Recently, parameter adaptation has been used in many context-aware systems [SG02, DA00, KSPR01], in which software execution is directly affected by the external environment. 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 [HS96, AC03, Ven02, CHS01, RBH+98, MPAS03]. In comparison to parameter adaptation, compositional adaptation enables an application to adopt new algorithms for addressing concerns unforeseen during original design and construction. The flexibility of compositional adaptation enables more than simple tuning of program variables or strategy selection. Dynamic recomposition is needed when resource limitations (for example, memory in small devices) restrict the number of responding components that can be deployed simultaneously, or when adding new behavior to deployed systems to accommodate unanticipated conditions or requirements (for example, detection of and response to a new security attack).

In its internal design, UCM employs both parameterized and compositional adaptation. Instead of reinventing the wheel, UCM incorporates existing adaptive and reflective middleware toolkits to provide self-management using only a high-level guideline from communication applications. ACE, Ensemble, and Open ORB are among the projects that we closely follow to incorporate some of their services inside UCM. Adaptive Communication Environment (ACE) [Sch93, SH02] is a real-time object-oriented framework written in C++ that wraps many OS services and provides a variety of communication-related patterns for use by distributed applications.

Ensemble [RBH+98] from Cornell University is a groupware communication toolkit that supports distributed applications with application-specific communication protocols. Central to the design is the construction of protocol stacks from fine-grained components, called micro-protocols. For example, to support QoS monitoring, Ensemble enables insertion of detectors in the protocol graph. These detectors can trigger dynamic adaptation by distributing a new protocol-graph specification to all involved participants using a reconfiguration protocol. Ensemble provides a number of reusable micro-protocols in its library, and new micro-protocols can also be developed and used in Ensemble. Similar to ACE, Ensemble provides a process-wide adaptation.

In the Adapt Project, Blair et al. [BCD97] investigated middleware implementation for mobile multimedia applications that can be dynamically adapted in response to the environmental changes. In the OpenORB project [BCRP98], the successor to the Adapt project, Blair et al. focused on the role of computational reflection in middleware.

Finally, in confronting a dynamic physical world, decision making in adaptive systems must modify software composition to better fit the current environment while preventing damage or loss of service. Decision makers must monitor both their physical and virtual environments using software and hardware sensors. Moreover, pervasive computing environments may require that software learn about and adapt to user behavior. Some existing decision makers use rule-based approaches [LJK+00], while others are supported by theoretical models, including those based on control theory [HS00], resource optimization [PSGS03], and those inspired by biological processes, such as the human nervous system [KC03] and emergent behavior in species that form colonies [WS00]. UCM currently employs a rule-based decision maker. In near future, we are planning to investigate the value of other decision makers in UCM.

6 Conclusion and Future work

We have proposed UCM, a unified high-level abstraction that isolates and separates the complexities of network-level communication control and media delivery from the application-dependent communication logic. We have identified the requirements of the UCM abstraction required for the class of user-centric multimedia communication applications. UCM facilitates rapid creation of portable communication applications. The design of UCM is based on an extensible and reusable software framework that provides a unified communication interface to applications with diverse communication logic and using various media types, by providing an encapsulation of heterogeneous network environments. In the future, we plan to enhance the extensibility, reusability, and self-management of UCM. . Also, the prototype will be improved with respect to performance and usability issues.

Acknowledgements We thank Nagarajan Prabakar, Peter Clarke, and Vagelis Hristidis for their participation in discussions and their contributions in the implementation of the UCM prototype.

Reference

[AC03] 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),(Providence, Rhode Island), May 2003.

[BCD97] G. Blair, G. Coulson, and N. Davies, “Adaptive middleware for mobile multimedia applications,” in Proceedings of the Eighth International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV), pp. 259–273, 1997.

[BCP+04] Gregory W. Bond, Eric Cheung, K. Hal Purdy, Pamela Zave, and J. Christopher Ramming, “An open architecture for next-generation telecommunication services”, ACM Transactions on Internet Technology IV(1):83-123, February 2004.

[BCRP98] G. S. Blair, G. Coulson, P. Robin, and M. Papathomas, “An architecture for next generation middleware,” in Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware’98), (The Lake District, England), September 1998.

[CHS01] 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), (Mesa, Arizona), pp. 635–643, April 2001.

[DA00] A. K. Dey and G. D. Abowd, “The context toolkit: Aiding the development of context-aware applications,” in Proceedings of the 22nd International Conference on Software Engineering (ICSE): Workshop on Software Engineering for Wearable and Pervasive Computing, (Limerick, Ireland), June 2000.

[DSC+04] Yi Deng, S. Masoud Sadjadi, Peter Clarke, Chi Zhang, Vagelis Hristidis, Raju Rangaswami, and Nagarajan Prabakar, "A Unified Architectural Model for On-Demand User-Centric Communications", Technical Report FIU-SCIS-2005-09, School of Computing and Information Sciences, Florida International University, 2004.

[FHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements od Reusable Object-Oriented Software. Addison-Wesley Professional Computing Series, New York, NY: Addison-Wesley Publishing Company,1995.

[GNCS04] Xiaohui Gu, Klara Nahrstedt, Rong Chang, Zon-Yin Shae, “An Overlay Based QoS-Aware Voice-Over-IP Conferencing System”, In Proceedings of IEEE International Conference on Multimedia and Expo (ICME2004), June 27-30, 2004.

[GRR00] N. Greene, M. Ramalho, and B. Rosen, “Media Gateway Control Protocol Architecture and Requirements”, RFC 2805, April 2000.

[H323] ITU-T Recommendation H.323v.4 "Packet-based multimedia communications systems", November 2000.

[HS96] 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.

[HS00] M. A. Hiltunen and R. D. Schlichting, “The Cactus approach to building configurable middleware services,” in Proceedings of the Workshop on Dependable System Middleware and Group Communication (DSMGC 2000), (Nuremberg, Germany), October 2000.

[HSSR99] M. Handley, H. Schulzrinne, E. Schooler and J. Rosenberg, “SIP: Session Initiation Protocol”, RFC 2543, March 1999.

[ICEB] The ICEBERG Project, University of California, Berkeley. http://iceberg.cs.berkeley.edu/

[JAIN] JAIN SIP, https://jain-sip.dev.java.net/

[JMF] Java Media Framework API, http://java.sun.com/products/java-media/jmf/

[JZ98] Michael Jackson and Pamela Zave, “Distributed feature composition: A virtual architecture for telecommunications services”, IEEE Transactions on Software Engineering XXIV(10):831-847, October 1998.

[KC03] J. O. Kephart and D. M. Chess, “The vision of autonomic computing,” IEEE Computer, vol. 36, pp. 41–50, January 2003.

[KR01] J. F. Kurose and K. W. Ross, Computer Networking: A Top-Down Approach Featuring the Internet. Boston, Massachusetts: AddisonWesley, 2001.

[KSPR01] G. Kortuem, J. Schneider, D. Preuitt, T. G. C. Thompson, S. Fickas, and Z. Segall, “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), (Link¨opings, Sweden), August 2001.

[LJK+00] B. Li, W. Jeon, W. Kalter, K. Nahrstedt, and J. Seo, “Adaptive middleware architecture for a distributed omni-directional visual tracking system,” in Proceedings of SPIE Multimedia Computing and Networking 2000 (MMCN’00), January 2000.

[LS03] Jonathan Lennox and Henning Schulzrinne, “A Protocol for Reliable Decentralized Conferencing”, ACM NOSSDAV 2003.

[MPAS03] 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 (Special Issue on Wireless Internet), pp. 713–726, June 2003.

[MSKC04] Philip K. McKinley, Masoud Sadjadi, Eric P. Kasten, and Betty H. C. Cheng, “Composing adaptive software”, IEEE Computer, pages 56-64, July 2004.

[PSGS03] V. Poladian, J. P. Sousa, D. Garlan, and M. Shaw, “Dynamic configuration of resource-aware services,” in Proceedings of the 26th International Conference on Software Engineering, (Edinburgh, Scotland), May 2004.

[RBH+98] R. van Renesse, K. P. Birman, M. Hayden, A. Vaysburd, and D. Karr, “Building adaptive systems using Ensemble,” Software Practice and Experience, vol. 28, p. 963979, August 1998.

[Ros05] J. Rosenberg, “A Framework for Conferencing with the Session Initiation Protocol”, Internet Draft, May 27, 2005. http://www.ietf.org/internet-drafts/draft-ietf-sipping-conferencing-framework-05.txt

[RWHM03] 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.

[SCFJ03] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, “RTP: A Transport Protocol for Real-Time Applications”, RFC 3550. July 2003.

[Sch93] 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.

[SG02] 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.

[SH02] D. C. Schmidt and S. D. Huston, C++ Network Programming: Mastering Complexity Using ACE and Patterns. Addison-Wesley Longman, 2002.

[Ven02] N. Venkatasubramanian, “Safe composability of middleware services,” Communications of the ACM, vol. 45, June 2002.

[WS00] M. Wang and T. Suda, “The bio-networking architecture: A biologically inspired approach to the design of scalable, adaptive, and urvivable/available network applications,” Tech. Rep. 00-03, Department of Information and Computer Science, Unversity of California, Irvine, California, February 2000.

[ZGS04] Pamela Zave, Healfdene H. Goguen, and Thomas M. Smith, “Component coordination: A telecommunication case study”, Computer Networks 45(5):645-664, August 2004.