

# A Unified Architectural Model for On-Demand User-Centric Communications

*Yi Deng, S. Masoud Sadjadi, Peter Clarke, Chi Zhang,  
Vagelis Hristidis, Raju Rangaswami, and Nagarajan Prabakar*

**Technical Report: FIU-SCIS-2005-09**

*School of Computing and Information Sciences  
Florida International University*

## **Abstract:**

The rapid growth of networking technologies has drastically changed the way we communicate and enabled a wide range of communication applications. However, these applications have been conceived, designed, and developed separately with little or no connection to each other, resulting in a fragmented and incompatible set of technologies and products. Building new communication applications requires a lengthy and costly development cycle, which severely limits the pace of innovation. Current applications are also typically incapable of responding to changes in user communication needs as well as changing network infrastructure and device technology. In this article, we address these issues and present the *Unified Communication Model* (UCM), a new and user-centric approach for conceiving, generating, and delivering communication applications on-demand. We also introduce a prototype design and implementation of UCM and discuss future research directions toward realizing next generation communication applications.

## **1. Introduction**















The convergence of data, voice, and multimedia communication over digital networks coupled with the continuous improvement in network capacity and reliability has enabled a wide range of communication applications<sup>1</sup>. Examples range from *general-purpose* communication applications such as a simple phone call, conference call, video conferencing, and instant messaging to more *specialized* applications such as disaster management and telemedicine. It is likely that the pace of innovation of new communication applications will accelerate further. However, the lengthy, costly, and inflexible development cycle for new applications hinders the realization of this vision. Current communication systems continue to be conceived, designed, and developed separately with little or no connection to each other. These applications are also typically incapable of accommodating changes in end-user communication needs, the dynamics of the underlying network, and new device and network technologies, without costly development cycle [Ker05].

The root cause to the above problems is that there is no common model, architecture, or















---

<sup>1</sup> We use the term “communication application” to refer to communication systems or tools targeted for end-users, ranging from systems supporting voice calls to domain specific systems supporting telemedicine.

systematic method of design for developing communication applications that are reconfigurable and adaptive. The stovepipe development of current systems dictates that they cannot promptly respond to the varying and changing needs of their end-users or to the changing dynamics of the underlying network or devices. These issues force the *end-users* to accommodate the limitations of technology and move from one tool to another to get their job done, rather than to adapt the *tools* to suit the users' needs. Figure 1 (a) illustrates the fragmented set of technologies and products that are built directly on top of low-level networking protocols and technologies as a result of the stovepipe development approach.

						
						
Phone Call	Conference Call App.	Video Conferencing	Instant Messaging	Distance Learning	Disaster Management	Telemedicine Application
IP Telephony Services	Conferencing Services	Video Conf. Services	Messaging Services	Dist. Learning Services	Disaster Mgmt. Services	Telemedicine Services
RTSP	SCP	SCTP	SIP	RTP/RTCP	HTTP	H.323
Internet2	PSTN	Wireless LAN	Cellular Network	Ad Hoc Network	Enterprise Network	Internet

(a) Current approach using stovepipe architecture.

						
						
Phone Call	Conference Call App.	Video Conferencing	Instant Messaging	Distance Learning	Disaster Management	Telemedicine Application
<b>Unified Communication Architecture (UCA)</b>						
RTSP	SCP	SCTP	SIP	RTP/RTCP	HTTP	H.323
Internet2	PSTN	Wireless LAN	Cellular Network	Ad Hoc Network	Enterprise Network	Internet

(b) Our new approach using a unified layered architecture.

**Figure 1:** Architecture of communication applications: (a) current approach, (b) our new approach.

In this paper, we present the *Unified Communication Model (UCM)* that addresses the above problems. As opposed to the stovepipe development, UCM introduces a common architecture across different communication applications, and a new method for conceiving, generating and delivering a wide range of communication services to end-users based on this architecture. Rather than developing communication applications from ground up, this new approach partitions the task of modeling user communication logic from the execution of such logic. A

similar practice of separating policy from mechanism has been popular in the operating systems community for several decades [LCC+75]. Further, UCM separates the control and coordination of user-level communication from the actual delivery of the communication by the underlying networks. Such separation of concerns results in a layered Unified Communication Architecture (UCA), which provides horizontal compartmentalization of communication tasks and system components that handle these tasks. Consequently, system components and communication protocols common to different applications can be identified, shared and called upon to serve different user communication needs without being hard coded into a stovepipe system. The resulting communication architecture is independent of the underlying networking infrastructure, the device technology and the applications as illustrated by Figure 1 (b).

UCM supports modeling of the user communication logic with the *user communication schema*, which describes user communication needs on demand (hence is user-centric). Such a user communication schema<sup>2</sup> is negotiated (among communicating parties) and transformed by an automated synthesis process into a *communication control script*, which in run-time controls and coordinates the user communication independent of the specific underlying network infrastructure and configuration. The automation of the synthesis process is achieved by reusing and composing available system components, which perform common communication tasks and can be accumulated and updated over time. The introduction of a Communication Virtual Machine (CVM) provides a uniform API, which abstracts the coordination of user-level communication from the actual delivery of the data by the underlying networks with varying protocols and configurations. Because of this separation of concerns, the behavior of user communication, dictated by the user communication schema, can be changed or reconfigured at run-time to accommodate the changing user communication needs. The concrete elements and process of UCM are discussed further in Section 3.

We argue that UCM represents a promising approach for rapid development and deployment of communication applications in a way that accommodates the rapid advances of networking technologies and infrastructure as well as the varying and growing end-user communication needs. It addresses many of the critical issues raised at the beginning of this section regarding the current stovepipe development approach. Furthermore, the use of UCM raises a number of interesting research issues, which will be discussed in the ensuing sections of this paper.

## 2. A Motivating Example

Let us consider the following scenario: Eric is a general practitioner who is examining one of his patients. He observes an unusual symptom, so he decides to call and consult with Mary, who is a specialist. During their conversation, Mary calls John, who is a researcher working in a medical laboratory, and asks him to join the conversation. This turns the two-way call into a conference

---

<sup>2</sup> As discussed in Section 3, common communication tasks can be packaged into predefined communication services without the user being aware of the model.

call. Eric then decides to share parts of his patient's record with Mary and John and show them some related images. This turns the voice conference into a multimedia telemedicine application. Note that the sharing of the patient's record must adhere to the health data exchange policies [DoH96].

Clearly, carrying out the above scenario is possible with today's technology. Referring to Figure 1 (a), Eric would first place a phone call to reach Mary. Next, assuming Mary's phone has conferencing capability, she switches to a conference call to include John in a three-way conversation. Otherwise, they have to use a conferencing application such as Yahoo! Messenger. Eric would then use a separate custom developed system for sharing the patient's record with Mary and John. In case either Mary or John does not have access to such a custom application, Eric may need to send the images via email or a file sharing application. In general, although such scenarios can be accommodated with today's technology, the users would either have to jump between different tools (*e.g.*, phone, email, file-sharing, messenger application), or to rely on custom-developed applications, which are typically expensive and rigidly designed.

In the next section, we show how such communication needs can be satisfied on-demand and with ease under the UCM approach.

### **3. Unified Communication Model (UCM)**

The Unified Communication Model is a novel approach for rapidly developing communication applications through specification and generation. In this section, we present a possible architecture for UCM that provides a separation of concerns in the development of communication applications. It is noteworthy that a different architecture may implement the UCM approach equally well.

There are four major tasks that are required to be performed to serve the users' communication needs:

- (1) Conceive and describe the users' communication needs or requirements. In the case of a voice phone call, it is for the user to pick up the phone and dial a number. In the case of a multimedia conferencing, it is to specify who the participants of the conference are and what kind of media or data are allowed to be exchanged. In the case of the telemedicine application outlined in Section 2, it also includes the policy that governs who can access which part(s) of the patient's medical record.
- (2) Transform the above user communication requirements into a sequence of commands or actions, which when executed will control and coordinate the flow of user communication as dictated by the requirements.
- (3) Provide a platform or environment in which the said sequence of commands can be executed to regulate the flow of communication.
- (4) Deliver the media or data among the communicating parties through a communication network or networks.

Today, these four major tasks are typically hard coded in a given communication system or application, which also predefines the way a user or users will use the system. Such a stovepipe approach of design is the root cause of the problems discussed in Section 1.

At the heart of UCM is a conceptual layered architecture, the *Unified Communication Architecture (UCA)*, which provides a clean separation and compartmentalization of these major concerns [BMR+98], as illustrated in Figure 2. UCA divides communication concerns into four major levels of abstraction, which contribute toward implementing the communication services. The four levels of abstraction correspond to the key components of UCA:

- (1) ***user communication interface***, which allows users to declaratively specify their communication needs and requirements in the form of a *user communication schema*;
- (2) ***communication schema synthesizer***, which provides the process and techniques to automatically transform and synthesize a user communication schema to an executable form called *communication control script*;
- (3) ***communication engine***, which executes the communication control script to manage and coordinate the delivery of communication services to users, independent of the underlying network configuration; and
- (4) ***communication virtual machine (CVM)***, which provides a network-independent API to the communication engine and works with the underlying network protocols to deliver the communication services.

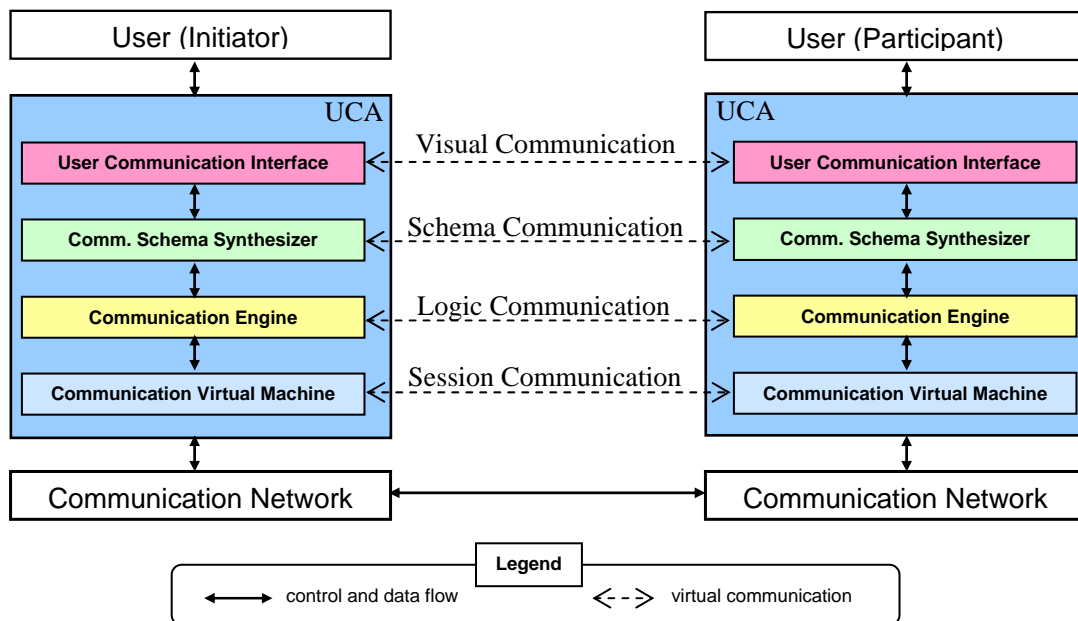


Figure 2: Scope of UCM and its layered architecture.

This layered division of responsibility is reminiscent of the OSI layered stack model for network communication [DZ95]. Each layer has a specific role in the stack and communicates logically with the peer-layer at a remote site during communication sessions. Each layer builds on the upper layers in the stack to finally realize the user-specified communication schema. Briefly, the *user communication interface* communicates visual changes to a schema instance to its peer layer; the *communication schema synthesizer* negotiates and communicates the schema with its peer layer; the *communication engine* communicates the actual communication logic defined in the schema instance with its peer; while the *communication virtual machine* handles session communication with its peer. We now explore the responsibilities of each layer in slightly more detail. A prototypical design of these layers, exposing a greater level of detail, is presented in Section 4.

**User Communication Interface.** The user communication interface is responsible, first, for providing users with the ability to define their communication schema, which describes the role of communicating parties and the overall user communication logic (*e.g.*, constraints, rules, and patterns). For this purpose, a *communication schema modeling language* is needed. Such a language should be simple and intuitive enough to support on-the-fly communication modeling without requiring knowledge of underlying networks and yet rich enough to describe a variety of communication logic. The language design poses many interesting research issues in its own right. We do not expect an end-user (*e.g.*, Eric in the scenario) to create a complex user communication schema (*e.g.*, the telemedicine schema), which is the job of domain experts or service providers. Second, the interface is responsible to check the validity and consistency of the communication schema as defined by the modeling language. Third, it must provide the user with a simple way of communicating and exchanging both media and data with other users (*e.g.*, through a graphical user interface) as well as display the current status of the communication to the user.

**Communication Schema Synthesizer.** The communication schema synthesizer performs several tasks. The first is schema negotiation among participants of the communication to ensure that all parties agree to a consistent schema. The schema negotiation process must be completed before the start of the actual communication. Second, the communication schema synthesizer *automatically* transforms the declarative user communication schema to an executable *communication control script*. This script represents the *network-independent* control logic for user-level communication sessions specified in the user communication schema and it defines and coordinates the delivery of services to users. It is critical that the communication schema synthesizer be fully automated and free of human intervention. To address the issue of automation, the communication schema synthesizer uses a repository containing pre-defined components for common as well as domain-specific communication. The communication schema synthesizer puts together the communication control script by appropriately combining pre-defined components (*e.g.*, for communication session establishment) based on the user

communication schema. Consequently, the capability of a schema synthesizer can be built up incrementally as the “middleware” components are developed. Third, the communication schema synthesizer is responsible for deploying (and possibly re-deploying) the control script to the communication engine. The design of automated and efficient synthesis techniques and the middleware components represents another class of interesting research issues.

**Communication Engine.** The communication engine is responsible, first, for executing the communication control script. Based on the communication logic in the control script, the communication engine invokes the common services provided by the CVM layer (described next) to perform several tasks including: (1) session creation, (2) adding a participant to the session, (3) adding a media to the session, (4) transmitting media, and (5) adjusting media QOS. Second, the communication engine is responsible for updating the user communication schema as a result of changes made by other parties in a communication session or dynamic network conditions. These changes are received in the form of callbacks from the CVM layer that may include: (1) session invitation, (2) receive media, (3) end media transmission, and (4) connection failed. Third, the communication engine is responsible for providing a safe state transition from a running communication control script to an updated control script that reflects either changed user communication needs or CVM callbacks. For example, when an end-user changes the communication schema, e.g. change a person-to-person call to a multi-way conference, in the middle of a session, the communication schema synthesizer will generate a new communication control script that reflects the user changes. Once the new communication control script is deployed to the communication engine, it should transfer the state of the old control script to the new one seamlessly and safely [Ven02].

**Communication Virtual Machine (CVM).** The CVM is responsible, first, for providing a unified *high-level* network-independent communication service (e.g., establish audio communication and then transfer a file to the session participants) to diverse communication applications. Second, the CVM is responsible for utilizing and coordinating the heterogeneous *low-level* networking functions (e.g., conduct signaling, encoding/decoding, and transmitting/receiving) provided by the underlying networks, systems, and libraries. Third, the CVM must exhibit a *self-managing* behavior that can respond to dynamics of the underlying device and network infrastructure. In essence, the CVM provides a uniform horizontal abstraction that separates and isolates the complexities of network-level communication control and media delivery from the complexity of user-oriented communication logic. Given the variety and complexity of current network infrastructure and configurations, the concept of CVM offers a novel approach to simplify application development and interoperation, and introduces many important research issues, e.g. self-management, dynamic configuration, definition of application independent communication API, software framework for hiding network heterogeneity, etc.

Together, the above layers cooperate to fulfill the promise of UCM – that of generating communication applications that are reconfigurable, adaptive, and flexible based only on a high-

level description of communication requirements. A summary of the high-level responsibilities assigned to each of these layers is presented in Table 1.

**Table 1:** A summary of the high-level tasks carried out at each layer of UCM.

UCA Layers	Tasks
<b>User Communication Interface</b>	<ol style="list-style-type: none"> <li>1. Create/modify the user communication schema based on user input.</li> <li>2. Check the correctness of the user communication schema.</li> <li>3. Handle user requirements of communication at run-time.</li> </ol>
<b>Communication Schema Synthesizer</b>	<ol style="list-style-type: none"> <li>1. Ensure the consistency of user communication schema through schema negotiation.</li> <li>2. Perform schema synthesis to obtain the communication control script.</li> <li>3. Deploy the script to the communication engine.</li> </ol>
<b>Communication Engine</b>	<ol style="list-style-type: none"> <li>1. Execute the communication control script.</li> <li>2. Update the user communication schema based on changes made by other participants.</li> <li>3. Perform a safe state transition from an older schema to an updated one.</li> </ol>
<b>Communication Virtual Machine</b>	<ol style="list-style-type: none"> <li>1. Provide a high-level communication API, which is independent of the platform.</li> <li>2. Utilize and coordinate the available, low-level network and hardware services.</li> <li>3. Provide self-management in response to dynamics of the underlying infrastructure.</li> </ol>

The UCM approach shares some common traits with the concept of model-driven software development [Bet04] which has found only limited success to date. In contrast to general-purpose, model-driven development, automatic generation of communication services is feasible in UCM for two reasons. First, UCM is restricted to the scope of communication services and does not bear the complexity of generating general-purpose applications. The complexity of communication logic can be carefully regulated through the design of the schema modeling language. Second, UCM utilizes communication middleware components (*e.g.*, those of ACE [SH02]) and server-side architectures (*e.g.*, [BCP+04]) as building blocks to generate communication applications. Such existing components encapsulate procedures, patterns, and algorithms governing basic communication services (*e.g.*, session establishment of person-to-person voice call, transmission of an image file, and real-time video streaming), which are well defined and well understood. The role of UCM is limited only to the identification and composition of such components [MSK+04].

#### 4. Prototypical Design and Implementation

This section discusses a prototypical design and implementation of UCM, which closely follows UCA. In this prototype, we identify one component corresponding to each layer of the UCA. We adopted web services technologies as the interfacing mechanism between these components for two reasons: (1) using web services provides the flexibility of using different programming languages for implementing the four components; and (2) it allows easy elimination of some components for resource-restricted devices (*e.g.*, PDAs) by following a client-server architecture, that is, we may decide to deploy only a subset of components to the device and deploy the rest on a remote server/proxy.

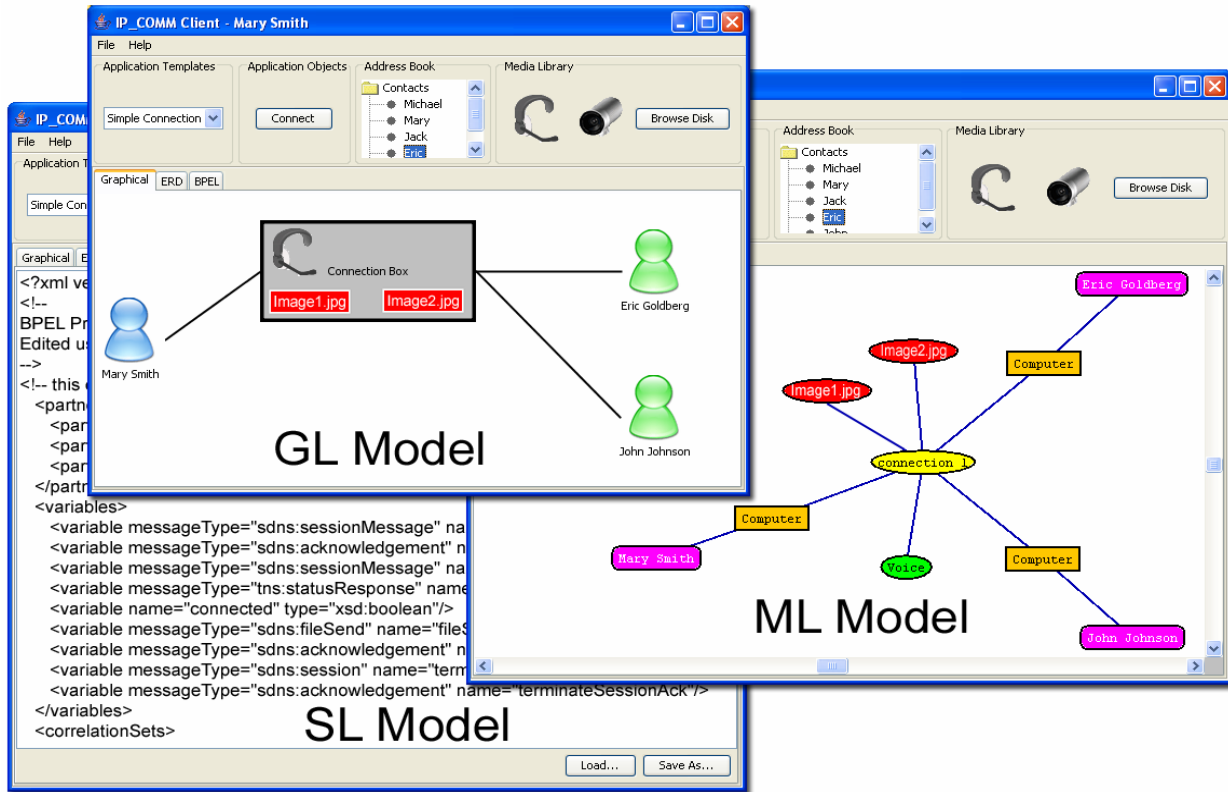


**User Communication Interface Prototype Component.** We define a user communication schema using three representations, collectively called communication *schema modeling language*, which map to or complement each other as explained below. These representations use the following languages respectively: (1) *Graphical Language (GL)*, which is a high level iconic language for describing the elements and their composition in a user communication schema, (2) *Modeling Language (ML)*, which is based on the Entity-Relationship model [Chen76] and formally describes the relationships and interactions between participating entities, and (3) *Scripting Language (SL)*, which is an XML-based language used to describe flow patterns in user communication.

The ML represents a superset of the information in the GL model. Additional information may include the devices that each party uses and their capabilities. The ML model uses *entities* and *relationships*. Entities are “static” components such as devices, persons, and data types. Relationships associate the entities in a communication session, such as “attached” and “connection”. The SL model may consist of pieces of code, written by a domain-expert developer, describing domain-specific (as opposed to generic) communication logic associated with communication components (the relationships of ML). These are then used by the communication schema synthesizer to generate the communication control script (also represented in SL). We chose BPEL [IBM03] as the SL since BPEL is widely accepted by the industry and is suitable for describing the user communication logic using its rich flow structures. The user communication interface was developed in Java.

Figure 3 shows the three representations (models) for the simple conference call in the scenario described in Section 2. Mary loads the GL model from the schema repository for a conference call. For this, Mary first selects the *Application Template* – “Simple communication”, next clicks on the *Application Object* – “Connect”, and finally selects the two participants (Eric and John) from her *Address Book*. The resources used in the connection are selected from the *Media Library* – “Audio” (shown as the headset), and the two JPG files (“Image1.jpg” and “Image2.jpg”), are dragged into the Connection Box by Eric.

As mentioned before, the user communication interface is also responsible for checking the correctness of the schema. This process involves checking the syntax and semantics of the user communication schema as represented in the SL model. In our prototype, we provide the syntax check using an XML Schema defined for our SL language. However, we do not yet provide a semantic check. In case a schema validation fails, an error message is conveyed to the user pointing out the error in the SL model. In our prototype, we assume that the origin of all schemas that are synthesized by UCM is the user communication interface; therefore, there is no need to check the schema for correctness in the communication schema synthesizer.



**Figure 3:** Example of the three representation models (GL, ML, and SL) in our prototype.

**Communication Schema Synthesizer Prototype Component.** The role of the communication schema synthesizer is to convert the user schema in SL representation to an executable communication control script (also in SL). To do so, we identify the following tasks to be performed by the communication schema synthesizer in sequence. First, it establishes a default communication session with the other participants for schema negotiation. Second, it performs *schema negotiation* to obtain the final schema in SL notation. This does not necessarily mean that the final schema for all the participants in a communication session are equivalent as each participant may have a different view of the schema (e.g., Eric may be consulting with another specialist at the same time and Mary may not be aware of such communication). Third, it splits the SL representation into the basic logic components, which are the relationships. For each relationship, the communication schema synthesizer retrieves the corresponding SL code from the communication schema repository and populates them with the parameters defined in the schema instance. It then populates the parameterized SL code pieces with additional device capability parameters. Finally, it glues the SL code pieces to obtain the communication control script.

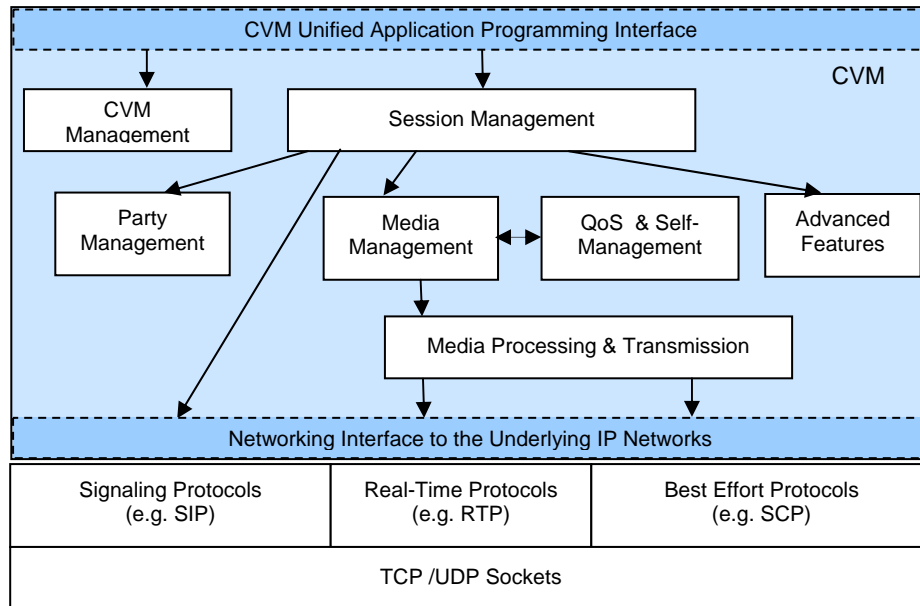
Our prototype for the communication schema synthesizer does not yet perform all the tasks mentioned above. Currently, it takes the SL representation of the communication schema as input to synthesize the communication control script from pieces of BPEL [IBM03] code created using ActiveWebflow.

**Communication Engine Prototype Component.** The main function of the communication engine is to execute the communication control script, which interacts with the communication schema synthesizer and the CVM. For our prototype, since BPEL is selected as the SL, we selected the Active BPEL engine, which is a BPEL engine by Active Endpoints as our execution engine. The Active BPEL engine is where the synthesized BPEL processes representing the communication schema are deployed. The deployed processes communicate with the CVM through a well-defined CVM interface. Note that safe state transition capability in communication engine is not yet available in this version of the prototype.

**Communication Virtual Machine (CVM) Prototype Component.** We define the internal architecture of the CVM based on the concept of *session*. A session is a communication process that involves a number of participants, who can be added or removed dynamically. Each participant of a session can multicast various media to all the other participants. A prototypical CVM architecture is outlined in Figure 4, which consists of three major aspects: the *Unified CVM API* is an application-independent and network-independent interface to the upper layer, through which high-level communication tasks can be specified; the *CVM Core* translates a high-level communication task into a series of operations that control and coordinate the underlying networking facilities; the *Networking Interface to the Underlying IP Networks* encapsulates and abstracts the heterogeneity of the network protocols and their interfaces. The CVM core is complex and further includes modules such as *Session Management*, *Participant Management*, *Media Management*, and *QoS and Self-Management*. The current prototype implementation utilizes the JAIN SIP and the JMF library, and supports SIP and RTP as underlying networking protocols.

## 5. Future Directions

UCM introduces a promising approach for rapidly conceiving, synthesizing and delivering communications services across different application domains on-demand. It is user-centric and dynamically configurable. It is network and device independent. These features make the model highly flexible and adaptive. The UCM approach also introduces a wide range of interesting and exciting new research issues, a subset of which are discussed in this paper. The results presented in this paper represent only an early study of the model and the approach it represents.



**Figure 4:** The internal architecture of CVM.

To capture the full potential of UCM, several open research issues need to be addressed. In particular, the scope of automated synthesis, though promising, still requires extensive study. We were successful in automating the generation of simple communication applications. However, the synthesis of applications with more complex business logic needs further investigation. A more powerful synthesis mechanism depends on a well designed communication schema modeling language. The language samples presented in this paper represents only its initial form. In addition, we are using an off-the-shelf Active BPEL engine as the communication engine in our prototype which does not address safe transition of communication state. We plan to address these issues in the near future.

To widen the scope of UCM, security mechanisms need to be incorporated that can provide privacy, authentication and/or access control for session initiation, authorization control in accessing resources, and data transfer encryption. Further, a set of quantitative measures is essential to monitor quality of service and to ensure the reliability of the overall system and each communication session. For instance, a strategy to mitigate the impact of the unreliable underlying network will involve a smart combination of real-time and offline communication.

We believe that resolving these challenging issues will make UCM an effective approach for developing the next generation of communication applications.

## Acknowledgements

This work was supported in part by the National Science Foundation under grant HRD-0317692. We thank Eduardo Monteiro, Onyeka Ezenwoye, Weixiang Sun, and Yingbo Wang for their participation in discussions and their contributions in the implementation of the UCM prototype.

## References

- [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) pp:83-123, February 2004.
- [Bet04] Jorn Bettin, “Model-driven software development: An emerging paradigm for industrialised software asset development”, Technical report, SoftMetaWare, June 2004.
- [BMR+98] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal, “Pattern-Oriented Software Architecture: A System of Patterns”, Wiley, 1998.
- [Che76] P. P. Chen, “The entity-relationship model: Toward a unified view of data”, *ACM Trans. Database Syst.* 1, 1, 9–36, 1976.
- [DoH96] Department of Health. Health Insurance Portability and Accountability Act (HIPPA) <http://dchealth.dc.gov/hipaa/hipaaoverview.shtm> (June 2005).
- [DZ95] John D. Day and Hubert Zimmermann, “The OSI Reference Model”, Conformance testing methodologies and architectures for OSI protocols, IEEE Computer Society Press pp:38-44, 1995.
- [Ker05] David Krebs. “The Mobile Software Stack for Voice, Data , and Converged Handheld Devices”, Mobile and Wireless Practice Venture Development Corporation, April 2005.
- [LCC+75] R. Levin, E. Cohen, W. Corwin, F. Pollack, and W. Wulf, “Policy/Mechanism Separation in Hydra”, In *Proceedings of the 5th ACM Symposium on Operating Systems Principles (SOSP '75)*, pages 132–140, University of Texas at Austin, November 1975.
- [MSK+04] Philip K. McKinley, Masoud Sadjadi, Eric P. Kasten, and Betty H. C. Cheng. “Composing adaptive software”, *IEEE Computer*, pages 56-64, July 2004.
- [RSA78] R. L. Rivest, A. Shamir, L. Adleman, “A Method for Obtaining Digital Signatures and Public-key Cryptosystems”, *Communications of the ACM*, 1978.
- [SH02] Douglas C. Schmidt and Stephen D. Huston. “C++ Network Programming: Mastering Complexity Using ACE and Patterns”, Addison-Wesley Longman, 2002.
- [Ven02] N. Venkatasubramanian, "Safe ‘Composability’ of Middleware Services", *Comm. ACM*, June 2002, pp. 49-52.