

Automatic Generation of User-Centric Multimedia Communication Services

Raju Rangaswami, S. Masoud Sadjadi, Nagarajan Prabakar, Yi Deng
Florida International University, 11200 SW 8th Street, Miami FL, USA 33199
{raju, sadjadi, prabakar, deng}@cs.fiu.edu

Abstract

Multimedia communication services today are conceived, designed, and developed in isolation, following a stovepipe approach. This has resulted in a fragmented and incompatible set of technologies and products. Building new communication services requires a lengthy and costly development cycle, which severely limits the pace of innovation. In this paper, we address the fundamental problem of automating the development of multimedia communication services. We propose a new paradigm for creating such services through declarative specification and generation, rather than through traditional design and development. Further, the proposed paradigm pays special attention to how the end-user specifies his/her communication needs, an important requirement largely ignored in existing approaches.

1. Introduction

In recent years, we have witnessed a great increase in the number and variety of multimedia communication services that have been developed and deployed. Examples range from IP telephony, instant messaging, video conferencing, multimedia collaboration, to specialized communication applications for telemedicine, disaster management, and scientific collaboration. Given the ease of creation of multimedia data, the continuous improvements in network capacity and reliability, and the varying and changing communication needs of end-users, it is likely that the pace of innovation in multimedia communication services will accelerate further.

Although this trend presents a tremendous opportunity for technological growth and for improved end-user experience, current approaches for developing multimedia communication services are severely lacking in several respects. Multimedia communication services today are conceived,

designed and developed following a stovepipe (vertical) approach that has resulted in a fragmented set of incompatible tools, technologies, and products. In addition, there is limited separation between application logic, the communication requirements, underlying platform specifics, and the networking protocols and infrastructure. The fragmented development approach also poses a great challenge in providing integrated multimedia communication services. Users are forced to hop between tools to satisfy their communication needs since the current suite of multimedia communication tools are technology-centric and largely ignore user-specific communication needs.

We argue that the root causes for holding back rapid innovation in multimedia communication services are the lack of user-centric developmental approaches, combined with stovepipe implementations. In this paper, we investigate a new paradigm for developing and deploying multimedia communication services through specification and automatic generation, rather than through traditional design and development. This paradigm advocates a model-driven process for conceiving and delivering multimedia communication that is tailor-made to fit user or application needs. Both general-purpose and domain-specific communication needs are specified in a model, called *communication schema*, which is independent of device types and underlying network configuration. This model is instantiated, negotiated, synthesized, and executed, by a fully automated process, to satisfy user communication needs. Using this approach, multimedia communication services can be built within hours or days, rather than months or years as required by current development cycles.

The focus of this paper is the automatic generation of user-centric multimedia communication services from communication schema descriptions, which allows changing the communication schema (or requirements) as required by the user or application at run-time. We refer to our automatic service generator as the *synthesis engine*. In contrast to general-

purpose, model-driven application development [12], here we focus on automatic synthesis for multimedia communication. Our preliminary study suggests that automated synthesis is largely feasible at least for the functional aspects of the communication such as coordination of communication features and capabilities and media delivery.

The **contributions** of this paper are as follows:

1. We propose a new paradigm for developing multimedia communication services through a model-driven, specification and automatic generation process.
2. We explore the scope of the automatic synthesis, given a declarative specification of communication needs, and show synthesis is feasible for the domain of user-centric communication services.
3. We present the automatic synthesis process, including algorithms for communication schema population, schema negotiation, and code generation.
4. We evaluate our proposed approach using a concrete case study of multimedia communication used in telemedicine.

The rest of this paper is organized as follows. Section 2 provides an overview of the Communication Virtual Machine (CVM), a comprehensive architecture that enables the proposed paradigm. Section 3 presents context for the automated generation process. Section 4 details the actual synthesis process as carried out by the synthesis engine. In Section 5, we present an evaluation of the proposed approach using a prototype that we have developed. Section 6 presents related research and we make concluding remarks in Section 7.

2. Communication Virtual Machine

To better understand the role of Synthesis, we present it in the context of the Communication Virtual Machine (CVM [6]), a comprehensive architecture for realizing communication services using the paradigm of specification and automatic generation.

The design of CVM draws from the concepts of model-driven engineering [12] and middleware-based architecture [8, 11]. However, by focusing on the communication domain only, CVM avoids the pitfalls of many general-purpose methods and techniques for model-driven engineering that are overreaching and consequently less effective.

The model-driven communication mentioned above is supported by the CVM layered architecture (Figure 1). These layers are common to and shared by

different communication applications. This architecture separates and encapsulates major concerns of communication modeling, synthesis, coordination, and the actual delivery of the communication by the underlying network and devices, into self-contained compartments with clear interface and responsibility.

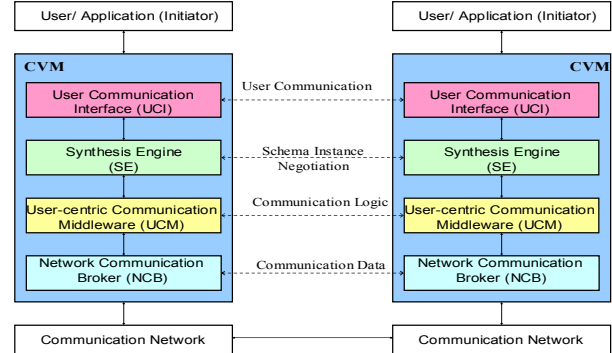


Figure 1. Layered CVM architecture

The CVM architecture divides the major communication tasks into four major levels of abstraction, which correspond to the four key components of CVM:

1. *User Communication Interface (UCI)*, which provides a language environment for users to specify their communication requirements in the form of a *user communication schema or schema instance*
2. *Synthesis Engine (SE)*, which is a suite of algorithms to automatically synthesize a user communication schema instance to an executable form called *communication control script*;
3. *User-centric Communication Middleware (UCM)*, which executes the communication control script to coordinate the delivery of communication services to users, independent of the underlying network configuration; and
4. *Network Communication Broker (NCB)*, which provides a network-independent API to UCM and works with the underlying network protocols to deliver the communication services.

The four layers collectively fulfill the promise of CVM – that of generating communication applications that are reconfigurable, adaptive, and flexible based only on a high-level description of communication requirements. In this paper we only focus on the automated schema synthesis process that transforms a declarative user communication schema instance (provided by the UCI) to an imperative control script (to be executed by the UCM).

3. The Context for Synthesis

In this section, we elaborate on two key concepts that help realize the CVM architecture when used in combination with the synthesis engine: *communication modeling language* and *user-centric communication middleware*. Note that these concepts are discussed elsewhere in detail [4,6]; we outline them here only for completeness.

3.1. Communication modeling language

The user-centric communication configuration and attributes (communication schema instance) are specified by the end user using a declarative communication modeling language (CML), in a graphical user environment [4]. This schema instance is represented as a modified ER diagram [3]. The UCI layer validates the user communication schema instance for syntax and semantics correctness and generates an XML specification of the communication schema instance that is processed by the synthesis engine. The UCI layer also processes call backs from the synthesis engine and notifies the user via the graphical user environment.

To illustrate the basic features of CML and the realization of the layered approach of the CVM, we present the following telemedicine communication scenario that takes place in an operating room. During a surgery of a patient Davis, Dr. Adams (surgeon) initiates a communication session with Dr. Brown (referring physician of Davis) and shares the echocardiogram and MRI images of the patient by interactively creating a communication schema instance with a voice activated browser. During this session, Dr. Brown wants to consult a cardiologist and includes Dr. Conway in the schema. Dr. Adams feeds a live video of the surgery to the

communication session and sends a structured vital sign chart (text data). This facilitates the discussion among all three and Dr. Adams performs successful surgery with the agreement of Dr. Brown and the consultation of Dr. Conway. Figure 2 presents a more easily interpretable graphical view of the same. Both views and corresponding interfaces for creation and modification are currently implemented in our prototype.

3.2. User-centric communication middleware

The UCM layer is responsible for executing the communication control script generated by the Synthesis Engine, for maintaining the states of user level communication (as opposed to network level), and for performing a safe state transition from a current running script to an updated one. The current state of a running script encapsulates its “program counter”, communication logs, data already exchanged, data in transition, and so on. In other words, UCM manages user communication sessions.

For the telemedicine communication scenario introduced above, the script generated by the synthesizer in response to the scenario initiated by Dr. Adams is as follows:

```
createSession("ID");
addParty("ID", "Dr. Brown");
addMedia("ID", "image", <URL>);
addMedia("ID", "audio", <URL>);
```

where the <URL>'s are replaced by their resolved values for the actual locations of the MRI imagery and echocardiogram of the patient. The above script is delivered to the UCM for execution and the actual delivery of communication will eventually be performed by the NCB layer.

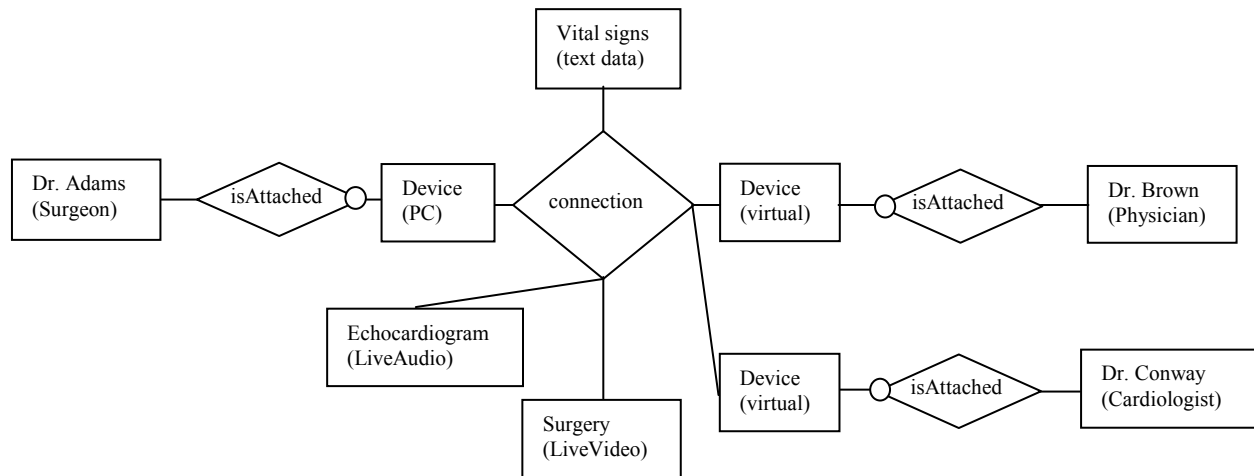


Figure 2. Graphical view of the schema instance (local view of Dr. Adams)

Finally, the UCM also provides feedback synthesis engine. For instance, it reports the change of session status to the SE so that necessary modifications can be made to the schema.

4. Synthesis Engine

The synthesis engine (SE) automatically transforms a declarative user communication schema (specified using the CML) to an imperative communication control script for deployment on the UCM. The SE is defined by its algorithms, processes, and techniques, which are used to generate the communication control scripts. These control scripts represent the network-independent control logic for user-level communication sessions.

The key challenge for the SE is complete automation, free of human intervention. With fully automatic synthesis, communication services can be generated from declarative user communication schema instance. The key question then is *whether such an automated transformation/synthesis is possible when automated program generation from declarative models for general-purpose systems is still beyond our reach?* We argue that for a limited subset of communication applications, such automation is feasible. Our preliminary study suggests that automated synthesis is largely feasible for the domain of user-centric multimedia communication services at least for the functional aspects of the communication such as coordination of communication requirements and capabilities as well as media delivery.

Given the role of the SE, we identify the following tasks it must perform:

1. Probe the local environment to align needs with communication capabilities and constraints and also determine the need for negotiation.
2. Ensure the consistency of user communication schema across participating end-points in a communication session.
3. Perform schema synthesis to obtain the communication control script to be deployed on the user-centric communication middleware.

Of the above tasks, #1 and #2 may involve handling exceptions and/or error conditions, which may require user-feedback for resolution. The rationale for designing the SE is to automate the handling of such exceptions and to employ user feedback only when unavoidable. We elaborate on this aspect in Sections 4.2 and 4.3.

The design of the SE follows a three-stage process: (1) *schema population*, during which the SE probes the environment to determine and account for local device communication capabilities and to

handle communication constraints, (2) *schema negotiation* among participants of communication, to determine the feasibility of the desired communication and to ensure that all parties agree to a consistent communication schema, and (3) *schema synthesis*, during which the SE determines the needs of communication and *automatically* transforms the schema to a *communication control script* deployable on a user-centric communication middleware.

Figure 3 depicts the architecture of the SE. The arrows depict control flow. We use the generic layered architecture proposed by Hill et al. [9] for the TinyOS platform, for interfacing with the CML (above) and UCM (below) layers. The *accepts* interface allows invocation of the SE with a communication schema instance. The SE interprets the schema, determining if there is a need for schema negotiation. If yes, the schema negotiation process is started; else, the SE proceeds to synthesize the schema instance, invoking the *uses* interface with the synthesized communication control script. If negotiation was invoked, the schema instance is first negotiated with all communication end-points (as specified in the schema instance) prior to synthesis. Finally, the SE contains an event handler for external negotiation requests, media progress/delivery notifications, and exceptions. Such handling may involve re-negotiation or user notification/feedback.

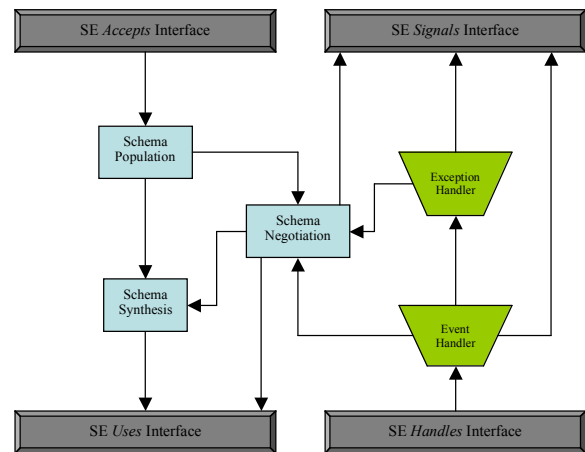


Figure 3. Synthesis Engine architecture.
Arrows depict control flow.

4.1. Schema population

The communication schema instance is a description of user communication needs in terms of the desired mode(s) of communication, remote participants, as well as the specific information that must be communicated. The first step in synthesizing the desired communication is schema population,

which probes the local environment to align communication needs with local device capabilities and constraints. Schema population also the need to negotiate communication parameters with remote participants involved in the communication.

Schema population augments the communication schema instance with the communication capabilities of the local device such as the media types supported, including specific format (or sub-type) information (e.g., real-media format of type video). The capability information is further enriched with type-specific information such as resolution and frame-rate of video or the bit-rate of audio supported. The populated schema instance is then aligned with the communication needs declared in the schema instance, employing user-feedback to resolve inconsistencies, if any. The populated schema information is also used in the schema negotiation stage (described next) to align the capabilities of all participants involved in the communication.

The need for schema negotiation arises the first time a communication is initiated and whenever there is a modification to the current communication schema instance. Specifically, it is required in the following scenarios: (a) the initiator of a new communication instantiates the corresponding schema with remote participant information for the first time, (b) a participant in an ongoing communication adds or deletes a participant, and (c) a participant in an ongoing communication adds/deletes a medium type to the current schema.

Addition or deletion of a participant requires renegotiation to inform other participants of the change as well as to accommodate the communication capabilities of the new set of participants. Addition or deletion of a medium type requires renegotiation to conform to the capabilities and preferences of the communicating participants. Note that the addition of a new instance of a medium-type (e.g. sending audio-file myfile.mp3) does not require renegotiation as this addition will occur only after the “audio” medium-type has been negotiated. No new capabilities are required of the end participants.

4.2. Schema negotiation

Schema negotiation is required to determine the feasibility of the desired communication and to ensure the consistency of the communication schema instance across the participating end-points in a user-communication session. A communication schema instance defined by user A may require a video connection to user B. However, if B’s device is not capable of video communication, this communication is not possible. Second, even if B’s device were

capable of video communication, B herself may not wish to engage in video communication with A at that time. In a multi-party communication scenario between A, B, and C, as initiated by A, C may not wish to communicate with B. Negotiation of the schema instance is required to understand the operating environment for the communication as well as to account for user preferences in communication. Apart from negotiating the initial schema instance before actual communication starts, schema *renegotiation* may also be required when a communication session is in progress.

We now describe our approach for schema negotiation used for negotiating the initial communication schema instance as well as performing renegotiation. Each participant in a communication session has a local copy of the schema instance, which they may change at runtime. Any change to the schema made locally may require an update to the local schema instances at all participating end-points. If two users in a session are simultaneously altering their schemas, concurrency problems arise. The synthesis engine uses a modified non-blocking three-phase commit protocol [15] for schema synchronization.¹

Each schema instance change initiates a negotiation process, which proceeds in three distinct phases. The final phase is the commit phase.

Phase I: The initiator reports the requested change to the schema instance to all remote participants, including any new participants being added, by sending the desired schema instance.

Phase II: The remote parties receive the changes and append their own *un-committed* changes, if any, to the schema instance. If this is the first time a schema instance is being negotiated or in case new participants are being added, the new participants also declare their device capabilities in the schema instance. Each remote participant sends this modified schema instance to the initiator.

Phase III: After the initiator receives the responses from all participants, all modifications from remote participants are merged. If the new schema instance differs from the original intent of the initiator, user feedback is employed to authorize the communication. The initiator then sends a final confirmation, either in the form of a consistent schema instance to be used for communication or to cancel the session.

¹ This option was favored over synchronization using a distributed lock, which disallows parallel schema instance modification. Further, distributed locking protocols are not robust to unstable network connections.

Since multiple parties may initiate schema negotiation simultaneously, negotiation requests from remote parties are queued together with the locally generated negotiation requests in a *synchronized negotiation request queue*. These requests are handled in order to ensure the consistency of the append operations described above.

4.3. Schema synthesis

As shown in Figure 3, the schema synthesis process is invoked either directly after schema population or after negotiation. Irrespective of the path taken, the schema synthesis process is the same. Its purpose is to transform the declarative communication schema instance into an imperative communication control script, executable on the user-centric communication middleware (see Section 3.2).

An XML schema for a communication session defines all device types and device instances that are part of the session, followed by the attributes of all participants, and the association between participants and device instances in the session. The synthesis algorithm is as follows:

1. Obtain the difference between the current XML schema instance and the previous (already synthesized) schema instance. If no previous schema instance exists, the entire new schema instance is used as the difference.
2. Augment the difference XML with context information including session ID and connection ID for each new entity (e.g., new participant or new medium instance), if absent. Now the difference XML is composed of one or more connection blocks.
3. Create an empty communication control script. For each connection block in the difference XML, (i) for each connection, if this connection did not exist in the previous version of the schema, add to the script a command to create a new session that implements this connection, (ii) for each participant, add to the script a command for adding a participant to the corresponding session, and (iii) for each medium instance, add to the script a command for adding the medium instance to the corresponding session.
4. Dispatch the communication control script to the UCM layer.

The XML schema of any communication session defines all devices, persons, and associations of the session, in sequence as a tree. As the synthesis algorithm processes the XML document as described above, it is evident that code for all features of the communication session will be generated.

4.4. Event handler

The synthesis engine architecture consists of an *event handler* (Figure 3) for handling event notifications from lower layers. These events can either be system notifications, exceptions, or error conditions. The event handler, dispatches the event to the appropriate handler. Remote negotiation requests are dispatched to the negotiation handler by adding them to the *synchronized negotiation request queue* (described in Section 4.2). Exception conditions such as loss of communication with a specific participant or temporary loss in network connectivity are dispatched to the *exception handler*, which may either initiate a re-negotiation request to handle the exception or intimate the user via the UCI layer if the exception cannot be handled internally due to schema instance-specific constraints. Finally, communication status updates such as the amount of progress in media delivery are directly notified to the UCI layer.

The synthesis engine delivers four types of notifications to the UCI layer. The `notifyMediaStatus` and `notifyParticipantStatus` signals notify the UCI about media delivery and participant connectivity. The `notifySIStatus` signal notifies the UCI about changes to the schema instance as a result of external changes due to other participants such as addition of new participant to an existing session or a change in capabilities of an existing participant, etc. Finally, the `notifyException` signals the UCI about exceptions such as lost network connection, when it cannot be handled internally.

5. Evaluation

To evaluate the effectiveness of the synthesis engine, we incorporated a prototype of synthesis engine into the CVM implementation [6]. The CVM uses the Opera 8.5, a voice-enabled browser, that enables creation, modification, and use of communication schema instance using voice commands. The synthesis engine prototype and the lower layers are implemented in Java, deployed on each node. JAIN SIP and Java Media Framework (JMF) are used for control and data communications, respectively. The current prototype propagates communication errors to the user. A future extension would be to add internal handling (obliviously to the user) of a subset of exceptions.

5.1. Reduced development time

We now obtain an estimate of the reduction in development time (and consequently, development cost) using the automatic synthesis approach. To do so, we used the open source Jabber chat application,

and also ourselves developed two Java-based applications that provided person-to-person voice call, and person-to-person video communication service respectively. These implementations used JMF and JAIN-SIP technologies. Table 2 summarizes these applications, their code sizes in lines of code (loc), their estimated development time using the traditional approach, and also shows the approximate specification and synthesis time for generating these applications using the CVM prototype. To estimate the development time by a trained programmer, we used the study of Ferguson et al. [7], whose findings reveal approximately 2500 lines of code per month per programmer. These numbers demonstrate the significance of our approach. Service creation time is reduced by several orders of magnitude. Even if we assume that only 25% of the code contributes to the functional aspects of the software, improvements in development time are still over two orders of magnitude. Further, the automated process introduces fewer bugs into the code-base, improving software reliability. This underlines the importance of using automated processes for synthesizing communication applications rather than follow traditional design and development.

5.2. Synthesis engine

To evaluate the time required for the actual synthesis process, we deployed CVM to 7 machines (desktops and laptops) in a combination of wired and wireless local area network. Ten demo users were created and used to represent 7 users communicating with each other

To verify the correctness of the synchronization protocol within the negotiation process, we initiated simultaneous modifications to the schema instance at different sites and verified the absence of any inconsistencies in the schema instances at the various end-points automatically over numerous iterations.

In addition, we instrumented the synthesis engine to obtain the time required to perform schema synthesis. The plot in Figure 5 shows the average time in seconds required for synthesis including the

negotiation/re negotiation stages when there are 2 to 7 participants present in a communication session. The results of these experiments show that the synthesis process scales linearly with the number of participants, and the process itself is dominated by the schema negotiation time. Higher numbers are not depicted due to lack of experimental infrastructure; however, we do not envision any issues limiting the linear scaling for larger participant-sets. The experiment demonstrates the practicality of schema synthesis process with a distributed negotiation algorithm; the negotiation time, which dominates the overall synthesis time incurs an acceptable delay.

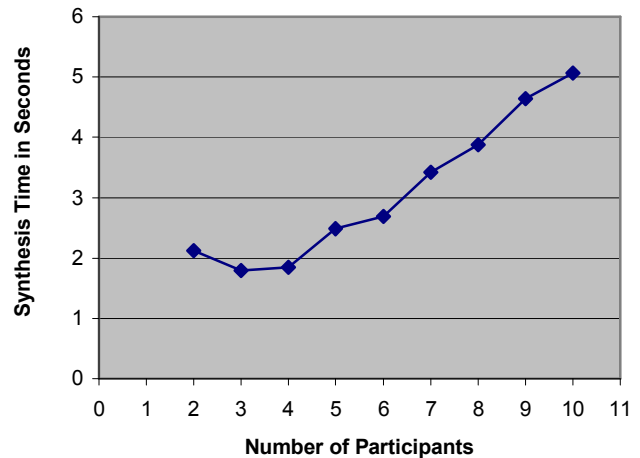


Figure 5: Average time required for negotiation.

6. Related Work

The CVM approach shares some common traits with the concept of model-driven engineering [2, 8, 12]. In contrast to general-purpose model-driven development, automatic generation of communication services is feasible in CVM for two reasons. First, CVM is restricted to the scope of communication services and does not bear the complexity of

Table 2. Reduced development time compared to traditional design and development.

Application Type	Application	loc	Est. development time	Spec/ Synthesis Time
Multi-user Text chat	Jabber-1.4.2	5528	2 months	< 5 minutes
Person-to-person voice call	Custom	9478	4 months	< 5 minutes
Person-to-person video comm.	Custom	16784	7 months	< 5 minutes

generating general-purpose applications. The complexity of communication logic can be carefully regulated through the design of the schema modeling language. Second, CVM utilizes communication middleware components (*e.g.*, those of ACE [14]) and server-side architectures (*e.g.*, [1]) 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 understood. The role of CVM is limited to the identification and composition of such components [10].

Heckel and Voigt [8] describe how models in UML are transformed into BPEL4WS using the concept of pair grammars. We use a similar approach in the UCI but our modeling language G-CML is far more restrictive than UML and hence far more manageable and its synthesis can be automated.

The work in [2] generates code from models using tool suites for specific application domains that were developed using a generic modeling environment. Examples of other approaches to code generation from higher-level specifications and languages include domain-specific languages, generative programming, generic programming, constraint languages, feature-oriented development, and aspect-oriented programming [5]. In our work, a generic SE generates control scripts from a CML description of communication logic, with restricted utility to the communication domain.

7. Conclusions

We have presented a new paradigm for providing user-centric multimedia communication services through declarative specification and automatic generation, rather than through design and development. Specification is simplified by identifying the key features of multimedia communication from a user's perspective, which are then captured in an XML-based communication modeling language (CML) definition. Our prototype further simplifies specification by providing a simple graphical interface. Automatic generation is made possible by a systematic three-step process of schema population, schema negotiation, and schema synthesis, combined with an event handler that allows the automatic handling of system events, exceptions,

and error conditions. The proposed paradigm enables the rapid creation of multimedia communication services, a core necessity in sustaining and improving the pace of innovation in this domain.

References

- [1] G. W. Bond, E. Cheung, K. H. Purdy, P. Zave, and J. C. Ramming, "An Open Architecture for Next-generation Telecommunication Services", *ACM Transactions on Internet Technology* IV(1) pp:83-123, February 2004.
- [2] K. Balasubramanian, A. Gokhale, G. Karsai, J. Sztipanovits, and S. Neema "Developing Applications Using Model-Driven Design Environments", *IEEE Computer*, pages 33 – 40, February 2006.
- [3] P. P. Chen. "The Entity-Relationship Model: Toward a Unified View of Data", *ACM Trans. Database Syst.* 1, 1, 9–36, 1976.
- [4] P. J. Clarke, V. Hristidis, Y. Wang, N. Prabakar and Y. Deng. "A Declarative Approach for Specifying User-Centric Communication", *Symposium on Collaborative Technologies and Systems (CTS)*, 2006.
- [5] K. Czarnecki and U. Eisenecker. "Generative Programming", Addison Wesley, 2000.
- [6] Y. Deng, M. Sadjadi, P. Clarke, C. Zhang, V. Hristidis, R. Rangaswami, and N. Prabakar. "A Communication Virtual Machine", *IEEE COMPSAC*, September 2006
- [7] P. Ferguson, W. S. Humphrey, S. Khajenoori, S. Macke, and A. "Matvya. Results of Applying the Personal Software Process ", *IEEE Computer* 30(5) pp 24-31, May 1997.
- [8] R. Heckel and H. Voigt. „Model-based Development of Executable Business Processes for Web Services”, *LNCS vol. 3098*, pages 559–584. Springer, Jun. 2004.
- [9] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, K. Pister. "System Architecture Directions for Networked Sensors", In *ASPLOS*, pp. 93-74, 2000.
- [10] P. K. McKinley, M. Sadjadi, E. P. Kasten, and B. H. C. Cheng. "Composing Adaptive Software", *IEEE Computer*, pages 56-64, July 2004.
- [11] D. C. Schmidt. "Middleware for Real-time and Embedded Systems", *Comm. of the ACM*, 45(6), June 2002.
- [12] D. C. Schmidt, "Model-Driven Engineering", *IEEE Computer*, February 2006, 25-31.
- [13] D. C. Schmidt. "Applying Patterns and Frameworks to Develop Object-Oriented Communication Software", volume 1 of *Handbook of Programming Languages*. MacMillan Computer Publishing, 1997.
- [14] D. C. Schmidt and S. D. Huston, "C++ Network Programming: Mastering Complexity Using ACE and Patterns", Addison-Wesley Longman, 2002.
- [15] D. Skeen, "Nonblocking Commit Protocols", pages 133-142, *SIGMOD* 1981.