Proceedings of the 11th IASTED International Conference
SOFTWARE ENGINEERING AND APPLICATIONS
November 19–21, 2007, Cambridge, MA, USA
ISBN Hardcopy: 978-0-88986-705-5 / CD: 978-0-88986-706-2

REALIZING COMMUNICATION SERVICES USING MODEL-DRIVEN DEVELOPMENT

Yingbo Wang, Peter J. Clarke, Yali Wu, Andrew A. Allen and Yi Deng
School of Computing and Information Sciences
Florida International University
Miami, FL 33199, USA
email: {ywang002, clarkep, ywu001, aalle004, deng}@cis.fiu.edu

ABSTRACT

The advances in technology to support complex communication services, such as the pervasiveness of mobile devices and the convergence of multimedia communication over digital networks, has resulted in a need for a new approach to model and realize communication services. The stovepipe approach used to develop today's communication applications is no longer effective since it results in a lengthy and costly development cycle. In this paper we present an approach that allows a user (end-user or domain expert) to model and realize communication services using a model-driven approach. We describe a semi-formal communication logic which is the meta-model for creating instances of user communication services. To show the applicability of our approach we demonstrate how models are created and realized using our prototype and a scenario from the healthcare domain.

KEY WORDS

Modeling-driven development, Communication services, Meta-Model.

1. Introduction

The pervasiveness of communication technology facilitates the widespread use of communication-intensive applications. Improvements in network capacity and reliability, as well as the wide use of communication devices such as PDAs and cell phones provide developers with the ability to create more complex communication-intensive applications. These applications cover domains such as telemedicine, disaster management, and scientific collaboration. The communication services used in these applications include various combinations of: IP telephony, instant messaging, video conferencing, and other forms of multimedia data transfers. However, traditional stovepipe approaches to developing communication-intensive applications are cumbersome and costly with long development cycles. Today, all trends indicate that the pace of innovation of new communication-intensive and collaborative applications is expected to accelerate even faster.

In this paper, we define a user-level communication logic meta-model to support the model-driven development of communication-intensive applications. This meta-model defines the communication primitives, constraints and association policies between these primitives. Using the communication logic meta-model the communication needs of different applications can be modeled during requirements analysis and be realized in days rather than weeks or months. The rapid realization of the model is achieved by automatically transforming this model into an executable script language that is then interpreted. The modeling, transformation and realization of communication services are made possible by using the concepts outlined in the Communication Virtual Machine (CVM) [3]. Finally, we present a prototype which works on top of the Skype platform [11] to illustrate the realization process.

The paper is organized as follows. Section 2 provides background on model-driven development. Section 3 defines the semi-formal model for communication logic. Section 4 describes the architecture of the prototype used to realize a communication model. Section 5 describes how the prototype creates and realizes communication models. Section 6 discusses the related work and we conclude in Section **??**.

2. Background

In this section we provide a generalized definitions of a model, artifact and meta-data. An overview of model-driven development is also presented.

2.1 Model Defiition

In this paper we use the generalized definition of a model as presented by Hailpern and Tarr [5]. A *model* M is defined as an abstraction over some part of a software product. M is semi-formally defined as a four tuple $M = < N, E, \Sigma_M, \Lambda_M >$, where N - is the set of model nodes, E - the set of directed model edges from node to node, Σ_M - an alphabet of model labels, and Λ_M - an annotation mapping function that maps either nodes or edges into labels ($\Lambda_M : N \cup E \rightarrow \Sigma_M$). Specialized models are denoted by M_K where K is an abbreviation representing the specialized model. For example, M_{UML} denotes an UML model.

An *artifact* (A_M) is a set of model elements of M that have a particular meaning in a specific domain or with respect to a particular problem solution. An artifact is therefore a subgraph of some model M. A *relationship* R maps artifacts in one model M_i to artifacts in another model M_j. R is therefore $< A_i, A_j, \Sigma_R, \Lambda_R >$, where A_i, A_j are the artifacts in models M_i, M_j, respectively; Σ_R are the labels in R assigned by Λ_R. Hailpern and Tarr [5] define *meta-data* as the set of annotations at both the model level, Λ_M, and relationship level, Λ_R. We use this definition of meta-data throughout the paper.

2.2 Model-Driven Development

Model-driven development (MDD) automates the transformation of models from one form to another [8]. The MDD process usually requires that there be a source model or a platform independent model(PIM), and a target model or platform specific model (PSM) [12]. The initial source model represents the concepts of a specific domain (M_{CS}), in our case user view of communication services, and the final target model contains the source code that interacts with the underlying communication infrastructure (M_{CI}).

To ensure the consistency of models during transformation the technique of *meta-modeling* is used [10]. We define the meta-models used to realize communication services based on the meta-data extracted from the communication logic models. Atkinson and Kühne [1] state that meta-modeling should consists of two orthogonal dimensions that support two forms of instantiation: *linguistic* - concerned with the language definition, and *ontological* - concerned with domain definition. We use both in defining the meta-models used in our approach. The ontological meta-modeling will be implemented using profiles and stereotypes provided in UML 2 [9].

3. Modeling Communication

As stated in Section 2.2 one of the essential properties of MDD is the automatic transformation between different models. Meta-models are essential to the transformation process and in this section we present an abstract model of communication that forms the basis of a meta-model. The meta-model also supports the notion of a *user* view, synonymous to a use case, and the *global* view synonymous to the application. We also use this meta-model to create a UML profile that is used in developing the graphical modeling environment.

3.1 Abstract Communication Model

We define a communication logic model (M_{CL}) as a four tuple consisting of a set of nodes (N_{CL}), edges (E_{CL}), an alphabet of model labels (Σ_{CL}) and an annotation mapping function (Λ_{CL}). We use sans serif font to denote literal labels. The communication logic model is recursively defined where the nodes are themselves models.

$$
\begin{array}{lcl}
N_{CL} & \in & \{M_P, M_C, M_I\} \\
E_{CL} & \in & \{<M_P, M_I>, <M_I, M_C>\} \\
\Sigma_{CL} & \in & \{attributes_{CL}, transmission, \mathsf{isAttached}\} \\
\Lambda_{CL} & = & \{M_P \rightarrow attributes_{CL}, \\
 & & \quad M_C \rightarrow attributes_{CL}, \\
 & & \quad M_I \rightarrow attributes_{CL}, \\
 & & \quad <M_P, M_I> \rightarrow \mathsf{isAttached}, \\
 & & \quad <M_I, M_C> \rightarrow transmission\}
\end{array}
$$

- M_P represents a Participant model. Each M_P can only send or receive data via an interface model. A M_P may be associated with one or more interface models.
- M_I represents an Interface model. An M_I is associated with one and only one M_P.
- M_C represents a Channel model. A M_C may be associated with one or more M_I. A M_I may also be associated with one or more M_C.
- $attributes_{CL}$ represents a set of attributes including a unique identifier.
- $transmission \in \{$send, receive, bi-directional, disconnected$\}$

A Participant model M_P is defined as follows:

$$
\begin{array}{lcl}
N_P & \in & \{Participant, M_{CL}\} \\
E_P & \in & \{\} \\
\Sigma_P & \in & \{attributes_P\} \\
\Lambda_P & = & \{Participant \rightarrow attributes_P, \\
 & & \quad M_{CL} \rightarrow attributes_P\}
\end{array}
$$

- A $Participant$ is a source or sink of data.
- $attributes_P$ represents a set of attributes for Participants.

An Interface model M_I is defined as follows:

$$
\begin{array}{lcl}
N_I & \in & \{Interface\} \\
E_I & \in & \{\} \\
\Sigma_I & \in & \{attributes_I\} \\
\Lambda_I & = & \{Interface \rightarrow attributes_I\}
\end{array}
$$

- An $Interface$ represents either the actual device or a virtual device used by the participant to pass data to the channel.
- $attributes_I$ represents a set of attributes for Interfaces.

A Channel model M_C is defined as follows:

$$
\begin{array}{lcl}
N_C & \in & \{Channel, M_D\} \\
E_C & \in & \{<Channel, M_D>\} \\
\Sigma_I & \in & \{attributes_C, \mathsf{allows}\} \\
\Lambda_I & = & \{Channel \rightarrow attributes_C, \\
 & & \quad M_D \rightarrow attributes_C, \\
 & & \quad <Channel, M_D> \rightarrow \mathsf{allows}\}
\end{array}
$$

- A $Channel$ is the conduit that allows data to pass between interfaces.
- M_D models the data transmitted through a channel.
- $attributes_C$ represents a set of attributes for the Channel model.

A Data model M_D is defined as follows:

$$\begin{array}{lcl} N_D & \in & \{M_M, M_F\} \\ E_D & \in & \{\} \\ \Sigma_D & \in & \{attributes_D\} \\ \Lambda_D & = & \{M_M \rightarrow attributes_D, M_F \rightarrow attributes_D\} \end{array}$$

- M_M represents a Medium model, i.e., model of the media types, e.g., audio video, text files and so on.
- M_F represents a Form model. A *Form* is a user defined type that allows the creation of complex data types.
- $attributes_D$ is a set of attributes for the Data model.

A Form model M_F is defined as follows:

$$\begin{array}{lcl} N_F & \in & \{M_M, M_F\} \\ E_F & \in & \{<M_F, M_M>, <M_F, M_F>\} \\ \Sigma_D & \in & \{attributes_F, \textsf{contains}\ \} \\ \Lambda_D & = & \{M_M \rightarrow attributes_F, M_F \rightarrow attributes_F, \\ & & \quad <M_F, M_M> \rightarrow \textsf{contains}, \\ & & \quad <M_F, M_F> \rightarrow \textsf{contains}\} \end{array}$$

- M_M represents a Medium model.
- $attributes_F$ represents a set of attributes for Form.

A Medium model M_M is defined as follows:

$$\begin{array}{lcl} N_M & \in & \{Medium\} \\ E_M & \in & \{\} \\ \Sigma_D & \in & \{attributes_M\} \\ \Lambda_D & = & \{Medium \rightarrow attributes_M\} \end{array}$$

- M_M represents a Medium model.
- $attributes_M$ represents a set of attributes for Medium.

3.2 Views of Communication

The semi-formal model defined in Section 3.1 represents the meta-model that can be used to generate a model for the complete structure of communication services provided to all the users for an application in some specified domain. An example of such an application would be a surgeon sharing patient data with the referring and attending doctors immediately after surgery. Since our aim is to model and realize communication services it is important to define different views of the communication logic model. Note that these views represent the aspects of communication in the use cases that define the application. For example, the surgeon and the doctors in the healthcare application each has a view of the communication. We use the notion of *artifact* introduced in Section 2.1 to define the view of a communication logic model.

A *view* of a communication logic model M_{CL} is an artifact generated by applying constraints to M_{CL}. It can therefore be stated that different views of the model M_{CL} are all subgraphs of M_{CL}. The constraints are applied to the edges, alphabet of model labels andthe annotation mapping function. In this paper we consider two views: (1) *Global view* - the communication logic model M_{CL}, and (2) *User view* - an artifact, $A_{M_{CL}}$, is a communication model of one user's perspective. We define the user view as:

1. Each participant model, M_P, is considered as a singleton.
2. There is a unique participant labeled as *local*.
3. All participants that can be reached from the local participant through exactly one channel are labeled as *remote*.
4. All channels in $A_{M_{CL}}$ are connected to the local participant through one interface.
5. Each participant is connected to a channel through an interface.
6. The only participants in $A_{M_{CL}}$ are labeled as local or remote.

The global view captures the requirements of a complete communication application. The user view captures communication requirements of one user's perspective. The prototype presented in this paper realizes the user view communication logic model. A user view can be projected from the global view by using a breadth-first traversal. The algorithm is:

1. Label the participant p as local representing the specified user.
2. Label all associated interface of p as local interface.
3. Label all channel associated with labeled interface. If a labeled interface has no associated channel, unlabel it.
4. Find all unlabeled interfaces associated with labeled channels. Label them as remote interface.
5. Label all dependent participants of remote interfaces as remote.

In order to construct a global view of the communication from several user views we assume that: (1) participant, interface and channel in different user views have unique identifier, and (2) input include views of all participants in the communication logic model. The main steps in the algorithm to merge all user views are as follows:

1. Set N_{global} as union of all users' N.
2. Set E_{global} as union of all users' E.
3. Create new labels and mapping functions to remove duplicate information in user's view.

3.3 UML Profile for Communication

In Section 3.1 we presented a semi-formal model for the communication logic that can be used to construct a global communication view consisting of a set of user communication views. The communication logic model is a meta-model and therefore can support the various transformations used during model-driven development. The abstract syntax and static semantics of this meta-model can therefore be used in tools to support the construction of valid

Stereotype	Base Class	Tagged Values	Constraints
`<<Participant>>`	`Class`	`id: String`	May only be associated with instances of `Interface`. `id` is unique.
`<<Interface>>`	`Class`	`id: String`	May be associated with at most one instance of `Participant`. `id` is unique.
`<<Channel>>`	`Class`	`id: String`	May only be associated with instances of `Interface` and `Data`. `id` is unique.
`<<Data>>`	`Class`		
`<<Local>>`	`Participant`		There is only one instance of `Local`
`<<Remote>>`	`Participant`		
`<<Device>>`	`Interface`		
`<<Medium>>`	`Data`		
`<<Form>>`	`Data`		
`<<isAttached>>`	`Association`		Instances of `Participant` are associated with instances of `Interface`
`<<allows>>`	`Association`		Instances of `Channel` are composed of instances of `Data`.
`<<contains>>`	`Association`		Instances of `Form` are composed of instances of `Medium`.
`<<connectTo>>`	`Association`		Instances of `Interface` are associated with instances of `Channel`.

Table 1. UML Profile for the user's view of a communication model

communication models. Given the fact that UML is a popular modeling language and UML artifacts can be imported into MDD development environments [14] we have created a UML 2 profile [9] for the user view of a communication model.

The UML 2 profile for the user view of a communication model is shown in Table 1. UML 2 profiles are specific kinds of packages that allows meta-models to be created for specialized domains. The UML profile in Table 1 consists of three kinds of artifacts: (1) stereotypes - define specific meta-classes, Column 1, (2) tagged values - define meta-attributes, i.e. attributes of the stereotype, Column 3, and (3) constraints - modeling guidelines i.e., restrictions on how the meta-model maybe used, Column 4. For example, in Row 1 the stereotype is `<< Participant>>` which is a base class, it contains the tagged value `id` of type string `String` and it may only contain variables of the class whose stereotype is `<<Interface>>`.

1. *userSchema* ::= *local connection* {*connection*}
2. *connection* ::= *mediaAttached* **connection** *remote* {*remote*}
3. *local* ::= **person isAttached** *device*
4. *remote* ::= *device* **isAttached person**
5. *mediaAttached* ::= {**medium**} {*form*}
6. *device* ::= **device deviceCapability** {**deviceCapability**}
7. *form* ::= {*form*} {**medium**} | **form**
8. **person** ::= personName$_A$ personID$_A$ personRole$_A$
9. **device** ::= deviceID$_A$
10. **medium** ::= builtinType$_A$ mediumURL$_A$ suggestedApplication$_A$ action$_A$
11. **deviceCapability** ::= builtinType$_A$
12. **form** ::= suggestedApplication$_A$ action$_A$
13. action$_A$ ::= "send" | "doNotSend" | "startApplication"

Figure 1. EBNF representation of X-CML.

4. Architecture of Prototype

The prototype developed to model and realize user-level communication services is based on the layered architecture of the Communication Virtual Machine (CVM) [3]. The CVM enables the realization of models created using the Communication Modeling Language (CML) [2]. The CVM consists of the following layers: (1) *user communication interface* (UCI), allows users to declaratively specify their communication needs and requirements (in CML), (2) *synthesis engine* (SE), generates an executable script from a CML model and negotiate the model with other participants in the communication, (3) *user-centric communication middleware* (UCM), executes the communication control script to manage and coordinate the delivery of communication services, and (4) *network communication broker* (NCB), which interfaces with the underlying networks to implement the communication services.

CML is a language generated from the user view of communication logic model. It is used to define a communication schema as well as a communication instance. A *communication schema*, or simply *schema*, defines the allowed configurations and data transfers. A schema is synonymous to a class in the object-oriented paradigm and an *instance* to an object. There are two equivalent variants of CML: the XML-based (X-CML) and the graphical (G-CML). Figure 1 shows a simplified version of the X-CML in EBNF form. The G-CML is used to create graphical communication models for both communication schemas and instances. The details of the CVM and CML can be found in [3] and [2], respectively.

Figure 2 shows the high-level architecture of the prototype. The top part of the figure shows the two possible options a user have to interact with the UCI. The first option allows the expert user/developer to interact with the prototype via the Communication Modeling Environment, where G-CML models are created. The second option is a user-friendly interface, external to the UCI, that allows the novice user to create communication models using an interface similar to that of an instant messenger application. In this paper we focus on model construction and transformation with the objectives of realizing communication ser-

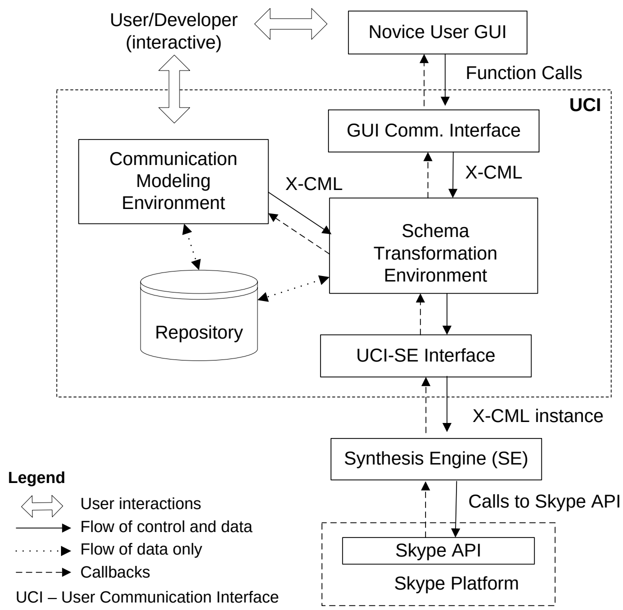

Figure 2. Architecture of the prototype.

vices. To this end the functionality of the lower layers of the CVM (UCM and NCB) are performed by Skype [11].

5. Realizing Communication Services

In this section we describe how models are created and realized using the implementation of the prototype presented in Section 4. A common scenario from the healthcare domain is used to illustrate how a developer can easily create an application that provides communication services to a doctor on-demand.

Scenario: After heart surgery Dr. Monteiro (the cardiologist) contacts Dr. Sanchez (family doctor) and Dr. Lopez (a heart specialist) to update them on a patient's condition. During communication with doctors Dr. Sanchez and Dr. Lopez, Dr. Monteiro sends them the post-surgery echocardiagram (echo) of the patient's heart and a text summary of the patient's current condition.

5.1 Model Creation

Communication models are created and validated in the UCI. The Communication Modeling Environment, shown in the upper left hand corner of the UCI, see Figure 2, is composed of (1) the graphical diagram editor used to create and validate the graphical models, and (2) the G-CML to X-CML transformer which converts the graphical model in G-CML to a text representation of the model in X-CML. The graphical diagram editor in the prototype was developed using a combination of the Graphical Modeling Framework (GMF) [15], the Eclipse Modeling Framework (EMF) [14] and the Graphical Editing Framework [13]. A summary of the steps used to create the graphical diagram editor are as follows:

1. Create a UML class diagram for the G-CML metamodel using a refined version of the UML profile in Table 1.
2. Use the class diagram in Step (1) to generate the Ecore model in EMF.
3. Through a series of transformations, the G-CML editor is generated from EMF [15]
4. The graphical diagram editor can now be used to create the model and output its XML representation.

The G-CML to X-CML transformer converts the XML representation of the G-CML model ($\textit{G-CML}_{XML}$) generated by the graphical diagram editor into the equivalent X-CML representation. The G-CML model maybe a schema or an instance of the required communication. An outline of the algorithm used to convert $\textit{G-CML}_{XML}$ to X-CML is as follows:

1. Parse the $\textit{G-CML}_{XML}$.
2. For each "connection" shape (c_g) in the parse tree of $\textit{G-CML}_{XML}$
   (a) create the new "connection" element (c_x) in X-CML using the XML schema [2]
   (b) Retrieve the shapes (S_g) directly linked to c_g
   (c) For each $s \in S_g$
      i. create a new element in X-CML (s_x)
      ii. add s_x as a child of c_x, where s_x maybe a "device", "medium", or "form"
3. For each shape s_g in the parse tree of $\textit{G-CML}_{XML}$ where s_g is a "person" or "isAttached"
   (a) create the corresponding X-CML element for s_g

The EBNF grammar shown in Figure 1 represents the XML schema that is used to construct the X-CML representation of the communication model.

5.2 Model Realization

Realizing a communication model requires two major steps, these are (1) checking the communication model to ensure it is a valid instance, and (2) translating the X-CML model into a communication control script containing calls to the underlying platform (Skype). The first step is performed by the *Schema Transformation Environment* in the UCI. In order to realize a communication model the Schema Transformation Environment loads either a schema or instance from the repository (or the Communication Modeling Environment) and checks if all the required attributes have values. This checking is done by a single traversal of the X-CML parse tree. If there are any required attribute values missing, usually true in the case of a schema, the user is requested to enter the missing values. For example, if the communication model is a schema for a two-party call then the user will be requested to enter the callee id of the remote participant.

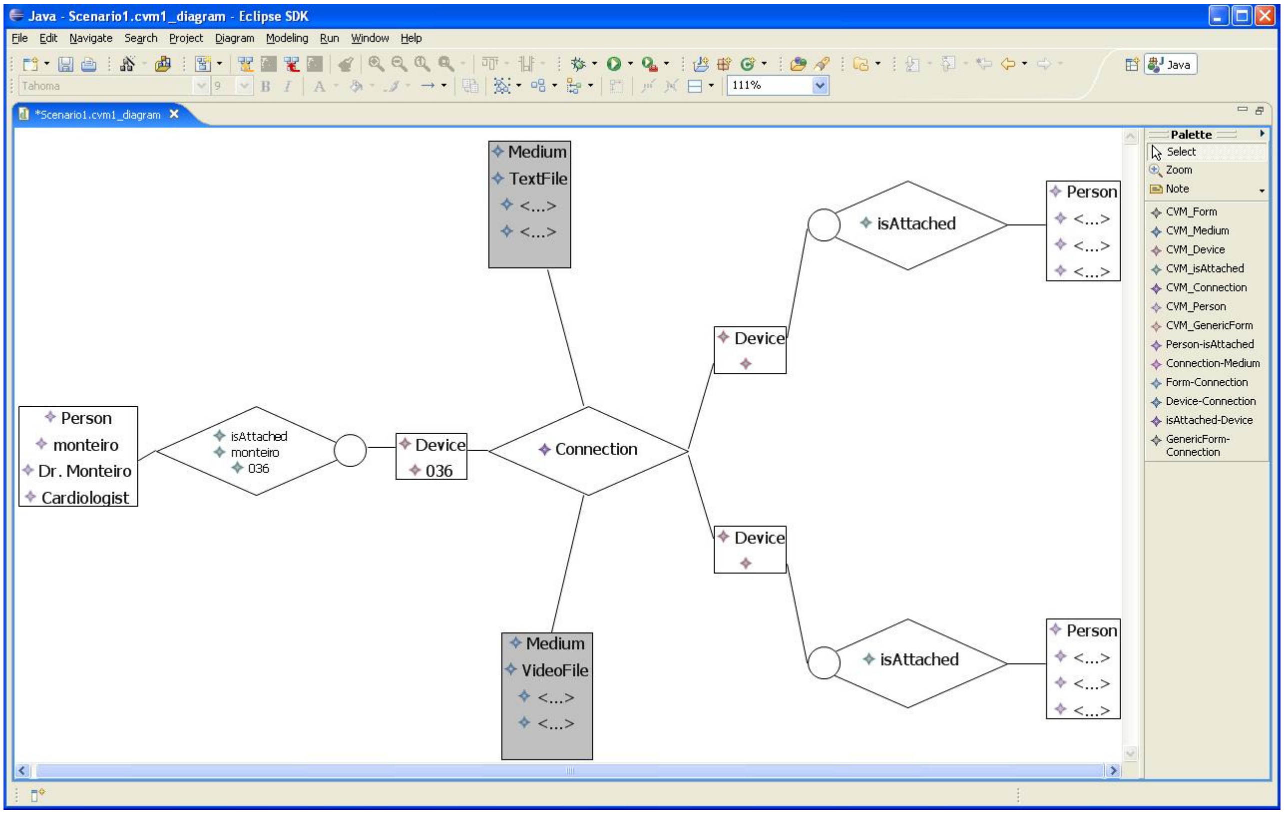

Figure 3. Screen shot of the communication schema.

Translating the X-CML model into a communication control script is done in the Synthesis Engine, as stated in Section 4. The following algorithm outlines how the X-CML is converted into a communication control script and executed for the initiation of communication.

1. Parse X-CML and traverse parse tree
2. For each connection c_i,
   (a) Identify the required media types associated with c_i
   (b) Establish a connection with the remote user(s) using a default type (e.g., audio)
   (c) If other media types are required then negotiate for each type. For example, if a video connection is required then listen for the remote video status. If the video is available then invoke the local video stream

Before a communication instance is realized the Skype application should be running and the caller logged in. Note that all participants in the communication should have Skype accounts.

6. Related Work

Our survey of the literature shows that although there has been much investigation done in domain specific modeling, there is currently no formal modeling techniques for modeling user-level communication services. Widespread modeling techniques currently used in software engineering and data modeling, like UML [9], are too generic and lack the formalism required for domain modeling, such as the modeling of user-level communication services. Greenfield et al. argue that although UML 2.0 is a useful modeling language, it is not an appropriate language for MDD [4]. Modeling and realizing communication aspects of applications for specific domains is still in its infancy.

Meanwhile, the software engineering community has been working on software frameworks for IP-based telecommunication (JAIN SIP [6], and Java Media Framework [7]). However, these communication services are usually tightly coupled with user applications. By applying the MDD [1] concepts, such as a visual modeling environment, meta-models, and model transformations, we decouple communication services from the application logic, hence providing a more effective way of modelling communication logic.

Deng et al. [3] introduced the Communication Virtual Machine (CVM) that provides the conceptual idea for the rapid realization of communication services that uses a layered architecture. However, there is no discussion on the theoretical foundations to support the automatic transformation of models between different layers of CVM. Clarke et al. [2] defined a simple and declarative Communication

Modeling Language (CML) for modeling user-level communication services. However, CML lacks a formal meta-model, which is the basis for any attempt at complete automation [12]. The meta-model in this paper provides a more complete and consistent approach to generating valid G-CML models and supports the automatic transformation between G-CML and X-CML. Using this meta-model, and the control script meta-model we can further automate the realization process.

7. Conclusion

In this paper we presented a semi-formal meta-model for user-level communication services and a prototype that realizes the communication aspects of an application using a model-driven approach. The prototype accesses the services of the underlying networks by using the Skype platform. Currently the realization process focuses on a user's view of the communication, that is the communication for an individual scenario. Our future work is to extend the construction and realization of the communication model, to include a global view. We also plan to investigate what network services can be provided by other open platform communication tools.

Acknowledgements

This work was supported in part by the National Science Foundation under grant HRD-0317692.

References

[1] Colin Atkinson and Thomas Kuhne. Model-driven development: A metamodeling foundation. *IEEE Softw.*, 20(5):36C41, 2003.
[2] Peter J. Clarke, Vagelis Hristidis, Yingbo Wang, Nagarajan Prabakar, and Yi Deng. A declarative approach for specifying user-centric communication. In *Proceeding of CTS 2006*, pages 89 C 98. IEEE, May 2006.
[3] Yi Deng, S. Masoud Sadjadi, Peter J. Clarke, Chi Zhang, Vagelis Hristidis, Raju Rangaswami, and Nagarajan Prabakar. A communication virtual machine. In *Proceeding of COMPSAC 06*, pages 521C531. IEEE Computer Society, 2006.
[4] J. Greenfield. Microsoft and domain specific languages. `http://blogs.msdn.com/jackgr/archive/2004/12/20/327726.aspx` (March 2007).
[5] B. Hailpern and P. Tarr. Model-driven development: the good, the bad, and the ugly. *IBM Syst. J.*, 45(3):451C461, 2006.
[6] JAIN-SIP.`https://jain-sip.dev.java.net/` (March 2006).
[7] Java Media Framework API. Internet2 working groups, and special in-terest groups, June 2005. `http://java.sun.com/products/java-media/jmf/` (May 2005).
[8] Stephen J. Mellor, Anthony N. Clark, and Takao Futagami. Guest editors introduction: Model-driven development. *IEEE Softw.*, 20(5):14C18, 2003.
[9] Object Management Group. Unified modeling language. `http://www.uml.org/` (Oct. 2006).
[10] Shane Sendall andWojtek Kozaczynski. Model transformation: The heart and soul of model-driven software development. *IEEE Softw.*, 20(5):42C45, 2003.
[11] Skype Limited. Skype developer zone, Feb. 2007. `https://developer.skype.com/`.
[12] Thomas Stahl, Markus Vlter, Jorn Bettin, Arno Haase, Simon Helsen, and Krzysztof Czarnecki. *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley and Sons, first edition, 2003.
[13] The Eclipse Foundation. Graphical editing framework (gef). `http://en.wikipedia.org/wiki/Eclipse_Modeling_Framework` (March 2007).
[14] Wikipedia. Eclipse modeling framework. `http://en.wikipedia.org/wiki/Eclipse_Modeling_Framework` (March 2007).
[15] Wikipedia. Graphical modeling framework. `http://en.wikipedia.org/wiki/Graphical_Modeling_Framework` (March 2007).