7-13-2011

A Domain Specific Modeling Approach for Coordinating User-Centric Communication Services

Yali Wu
Florida International University, ywu001@fiu.edu

DOI: 10.25148/etd.FI11081201

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

A DOMAIN SPECIFIC MODELING APPROACH FOR COORDINATING USER-CENTRIC COMMUNICATION SERVICES

A dissertation submitted in partial fulfillment of the

requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Yali Wu

2011

To: Dean Amir Mirmiran
College of Engineering and Computing

This dissertation, written by Yali Wu, and entitled A Domain Specific Modeling Approach for Coordinating User-Centric Communication Services, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this dissertation and recommend that it be approved.

S. Masoud Sadjadi

Xudong He

Vagelis Hristidis

Armando Barreto

Peter J. Clarke, Major Professor

Date of Defense: July 13, 2011

The dissertation of Yali Wu is approved.

Dean Amir Mirmiran
College of Engineering and Computing

Interim Dean Kevin O'Shea
University Graduate School

Florida International University, 2011

DEDICATION

To my mom, dad and friends, for your enduring support through this journey.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to all who contributed to the success of my PhD study and the completion of this thesis. I would especially like to thank the members of my committee, Drs. S. Masoud Sadjadi, Xudong He, Vagelis Hristidis and Armando Barreto, for their insightful comments, and thorough questioning, which helped me focus on my research ideas in completing my thesis. I am also grateful for Dr. Deng, the formal dean of the school, for his stimulating discussions with me, motivating research ideas and guidance during his service at FIU.

I would like to thank all students who have been working in the CVM team, including Yingbo Wang, Andrew Allen, Frank Hernandez, Juan Carlos Matinez, for their support and patience, and always willing to offer suggestions for improvement. To Professor Jean-Marc Jézéquel and fellow researchers of the University of Rennes I in France, and the great opportunity provided by the Partnership for International Research and Education (PIRE) program, for the generous support during my internship at Rennes, France in the summer of 2009.

Finally, I would like to express my deepest and foremost gratitude to my advisor, mentor and friend, Dr. Peter J. Clarke, for his guidance and continuous support of my PhD. study and research. Especially during my difficult times, when I am bewildered, confused and unsure about myself, it is Dr. Clarke who led me out of the way through constant encouragement. I thank him for his patience, perseverance, and hard support during my preparation for, and writing of this thesis.

ABSTRACT OF THE DISSERTATION

A DOMAIN SPECIFIC MODELING APPROACH FOR COORDINATING USER-CENTRIC COMMUNICATION SERVICES

by

Yali Wu

Florida International University, 2011

Miami, Florida

Professor Peter J. Clarke, Major Professor

Rapid advances in electronic communication devices and technologies have resulted in a shift in the way communication applications are being developed. These new development strategies provide abstract views of the underlying communication technologies and lead to the so-called *user-centric communication applications*. One user-centric communication (UCC) initiative is the Communication Virtual Machine (CVM) technology, which uses the Communication Modeling Language (CML) for modeling communication services and the CVM for realizing these services. In communication-intensive domains such as telemedicine and disaster management, there is an increasing need for user-centric communication applications that are domain-specific and that support the dynamic coordination of communication services commonly found in collaborative communication scenarios. However, UCC approaches like the CVM offer little support for the dynamic coordination of communication services resulting from inherent dependencies between individual steps of a collaboration task. Users either have to manually coordinate communication services, or reply on a process modeling technique to build customized solutions for services in a specific domain that are usually costly, rigidly defined and technology specific.

This dissertation proposes a domain-specific modeling approach to address this problem by extending the CVM technology with communication-specific abstractions of workflow concepts commonly found in business processes. The extension involves

(1) the definition of the Workflow Communication Modeling Language (WF-CML), a superset of CML, and (2) the extension of the functionality of CVM to process communication-specific workflows. The definition of WF-CML includes the meta-model and the dynamic semantics for control constructs and concurrency. We also extended the CVM prototype to handle the modeling and realization of WF-CML models. A comparative study of the proposed approach with other workflow environments validates the claimed benefits of WF-CML and CVM.

TABLE OF CONTENTS

LIST OF TABLES

TABLE PAGE

LIST OF FIGURES

FIGURE PAGE

CHAPTER 1

INTRODUCTION

Electronic communication has become pervasive in recent years. The advances in communication technologies and mobile devices (e.g. iPhone[2], iPad[3] and Android [29]) are dramatically expanding our communication capabilities and enabling a wide range of multimedia communication applications. Examples range from general-purpose applications that support voice calls, video conferencing, and instant messaging using a one-size-fits-all approach, to specialized applications in disaster management, distant learning and telemedicine [6, 16, 26, 42, 43]. The traditional stovepipe approach of developing communication intensive application binds the user-level communication logic with device types and underlying networks, results in rigid technology, limited utility, lengthy development cycle, and difficulty in integration. The challenges in creating flexible, interoperable communication services have led to new development strategies that move towards service convergence and user-centricity. These approaches provide abstract views of the underlying communication technologies and allow end-users to become more involved in the development of the so-called user-centric communication applications.

One of these user-centric communication initiative is the Communication Virtual Machine (CVM) technology, a new paradigm investigated by Deng et al.[17] for developing and realizing user-centric communication services to respond to increasing communication needs. The term user-centric is used to refer to those applications that provide services to the user, offer operating simplicity, and mask the complexity of the underlying technology [48]. The CVM technology consists of a domain-specific modeling language (DSML), the *Communication Modeling Language (CML)* [82], that is used to create communication models, and a semantic rich platform, the CVM, that

realized the created communication models. The time and cost of developing communication service can be largely reduced by using the CVM technology for formulating, synthesizing and executing new communication services.

However, user-centric communication approaches such as the CVM are limited to realizing individual communication scenarios. They only provide partial solutions for more elaborate domain-specific applications that require the composition and coordination of several individual communication scenarios to realize a collaborative process. In communication intensive domains like disaster management and healthcare, there is an increasing need for domain-specific communication applications that are user-driven and that support dynamic coordination of various communication services in a collaborative process [13, 22]. In such applications, collaboration and data sharing between geographically dispersed team members have inherent dependencies between individual steps of a collaboration task, and individual communication service instances need to be dynamically coordinated to ensure these dependencies. One scenario that typifies the need for this coordination is the patient discharge process, where the discharging physician may need to communicate with several other physicians, and to exchange various parts of the patient's electronic discharge package depending on specific healthcare information exchange rules. Another scenario might involve the coordination and communication between emergency and primary care physicians to treat patients in hospital emergency departments [22].

To realize these applications using existing techniques, users have to manually coordinate individual communication services in an ad-hoc manner, or rely on a general purpose process modeling technique to build customized solutions for services in a specific domain. For instance, modeling and realizing these scenarios using the CVM would require the scenario be modeled in several different communication models (CML models) and the users have to manually coordinate them. Customized

communication solutions could also be built on top of open communication APIs or application frameworks (e.g.,[35, 44]) and a process modeling techniques(e.g., CPL [51]) for specifying coordinated telephony services. However, the resulting applications are usually rigidly defined, technology-specific, costly, and inflexible when it comes to the adaptation of executing applications based on changing user preferences. The emergence of next generation networks has resulted in various user-centric service creation and delivery facilities [63, 58, 84]. While these approaches enable non-technically skilled users to create, manage and share their own convergent services, many of them lack the ability to coordinate individual communication services using the appropriate level of abstraction.

One way to address the above challenges of dynamically coordinating user-centric communication services is to raise the level of abstraction that user-centric communication services are coordinated through the use of model-driven engineering and domain-specific modeling techniques. In this dissertation, we investigate a domain-specific modeling language (DSML), Workflow Communication Modeling Language (WF-CML), that supports the rapid realization of coordinated user-centric communication services (UCCSs). WF-CML is an extension of the previously developed Communication Modeling Language (CML) [77] with communication-specific abstractions of workflow concepts commonly found in business processes. The current version of CML is limited to specifying structural specifications of the user's communication needs, leaving out the specification of control flow. Extending CML with a set of workflow abstractions introduces a natural solution for the required service coordination. Models created using WF-CML are interpreted using an extended implementation of the Communication Virtual Machine (CVM)[17], a run-time environment to dynamically synthesize, and execute UCCSs. The CVM is extended to coordinate the negotiation and media transfer processes based on events generated during the collaboration of UCCSs.

1.1 Motivation

To further motivate this research, we describe a detailed scenario from the healthcare domain. The authors have been collaborating with members of the Cardiology Division of Miami Children's Hospital (MCH) over the last 5 years to study the application of CVM in healthcare. One such scenario illustrates the process of discharging a patient involving discharge physician, senior clinician, primary care doctor, clinical pharmacist, attending physician and nurse practitioner. The associated flow of communication in this scenario is shown in Figure 1.1.

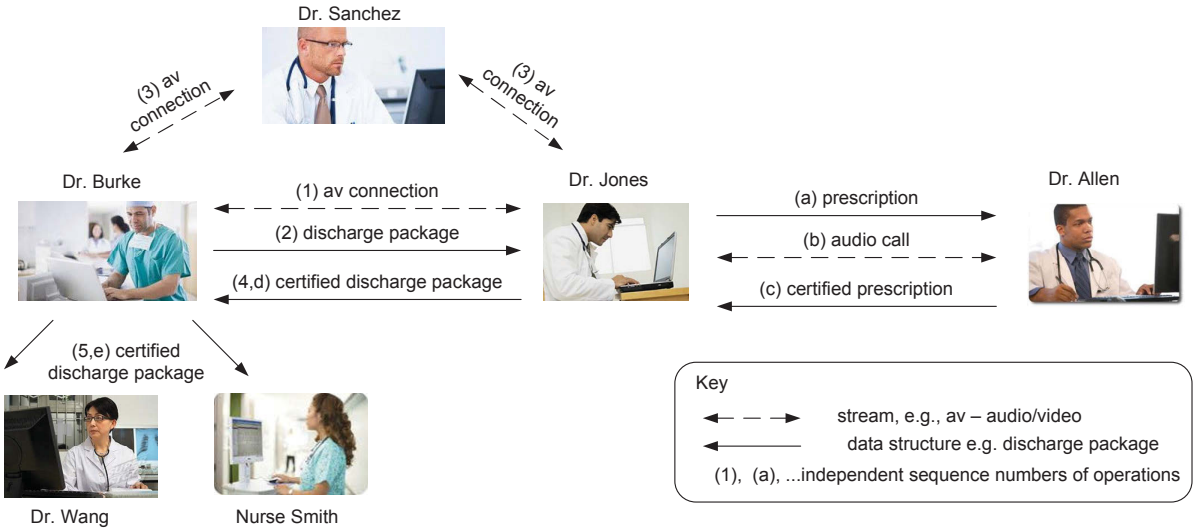

Figure 1.1: Sequence of Communication in the Patient Discharge Scenario.

Patient Discharge Scenario:

(1) On the day of discharge, Dr. Burke (DP - discharging physician) establishes an audio/video communication with Dr. Jones (SC - senior clinician) to discuss the discharge of baby Jane. During the conversation, Dr. Burke composes a discharge package and sends it to Dr. Jones to be certified. The package consists of a summary.pdf (non-stream file), summary of patient's condition and prescription; x-Ray of the patient's heart, heat_image.jpg(non-stream file); and a EcCard1.wmv(video

file), an echocardiogram (echo) of the patient's heart. After sending the package, Dr. Burke decides to contact Dr. Sanchez (PCP - primary care physician) to join the conversation with Dr. Jones to discuss the patient's condition.

(2) When Dr. Jones receives the request to certify the discharge package from Dr. Burke, Dr. Jones sends a request to Dr. Allen (CP - clinical pharmacist) to get the prescription certified i.e., check that the prescription meets specified hospital standards. The prescription is contained in the summary.pdf file. After Dr. Allen receives the request he initiates an audio communication with Dr. Jones to discuss the prescription. Dr. Allen then certifies the prescription and returns it to Dr. Jones.

(3) Upon receipt of the certified prescription Dr. Jones certifies the discharge package and sends it to Dr. Burke. Since the package is received within 24 hours and is certified, Dr. Burke then sends it to Nurse Smith (NP - nurse practitioner) and Dr. Wang (AP - attending physician).

If the package had not been certified and received within 24 hours, Dr. Burke would have sent an interim discharge note to Nurse Smith and Dr. Wang.

The scenario starts with a two-way communication between DP and SC, with the PCP being added to the call at a later time. It then includes the communication between SC and CP, followed by the communication between DP and AP and NP. The existing CVM technology captures the communication needs of this scenario in separate communication model instances [82], as shown in Figure 1.2. Figure 1.2(1)-(3) illustrates the three individual communication instances needed for this scenario. Each communication model specifies the structure of a communication scenario in a declarative manner (electronic media and form types to be exchanged among a set of participants, see Section 2.1.3 for more details). Dotted notations mean that

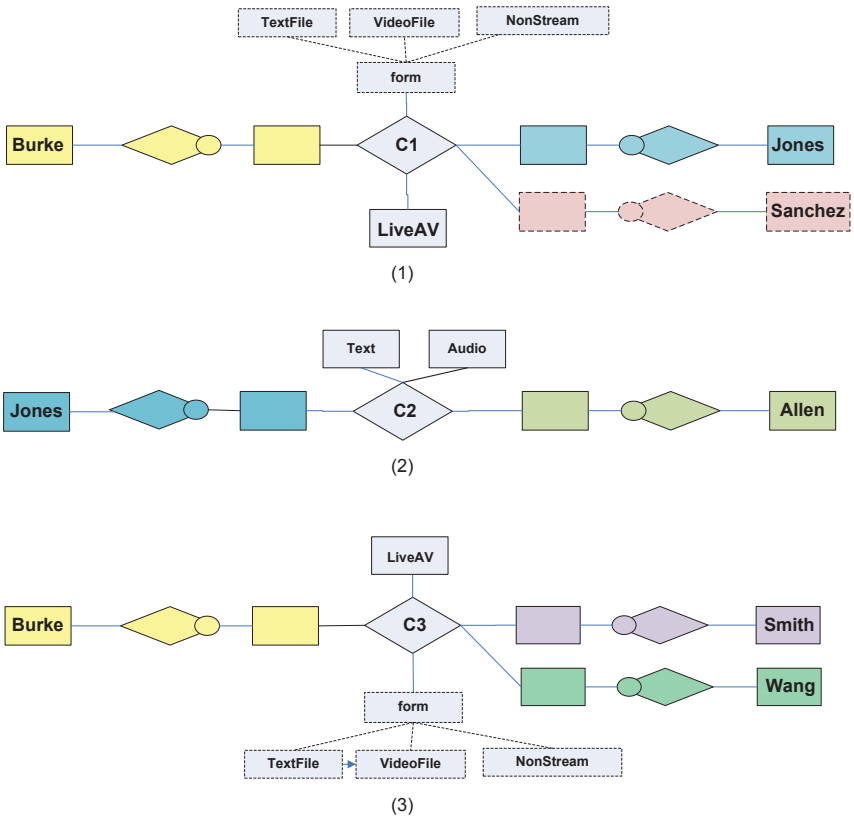

Figure 1.2: Communication Models for: (1) 3-way call between DP, SC and runtime added PCP (dotted notation) (2) 2-way call between SC and CP (3)3-way call between DP, NP and AP

the participant is added or information exchange initiated by the user during the conference at runtime.

To achieve a more efficient collaborative communication process, these different communication instances need to be coordinated during the execution of the scenario. The lack of support in existing approaches for dynamic coordination of communication services motivates us to introduce communication-specific abstractions of workflow concepts into communication models.

1.2 Problem Definition

We now introduce the main research problem of this dissertation: Despite the increasing complexity and high demand for coordinating user-centric communication services in a collaborative environment, currently there is no approach to quickly model and realize coordinated communication services in a way that is driven by end users and yet enables rapid realization.

To answer this question, we formulate the primary goal of this dissertation as: the design of an appropriate modeling abstraction for end users to model and realize coordinated user-centric communication services in a collaborative environment. Such a modeling abstraction should facilitate the specification and realization of communication services that require coordination.

We further decompose the goal into three specific objectives, therefore highlighting the specific contributions of this work:

First Subgoal: the design of a modeling language (WF-CML) for specifying coordinated user-centric communication services: Appropriate modeling abstraction for the coordination of user-centric communication should be designed that uses user-intuitive concepts and yet enables rapid realization.

Contribution: the syntactic design of WF-CML that includes a detailed domain analysis, a definition of its metamodel (abstract syntax and static semantics), and case studies to show how WF-CML is used to model and realize coordinated communication services.

Second Subgoal: the design of dynamic semantics for the modeling language (WF-CML) that enable dynamic synthesis and adaption for communication service coor-

dination. Semantics should be defined for the modeling language that facilitate the rapid realization of communication service involving workflows.

Contribution:: the definition of dynamic semantics for WF-CML that extends semantic specification of CML to incorporate control flow behavior required for service coordination. This extension includes the design of synthesis algorithms for analyzing WF-CML models, semi-formal definitions of a labeled transition system for managing the coordination logic, and a conceptual design of an execution architecture for WF-CML models. To evaluate the dynamic semantics, we use Kermeta [69] to perform simulations of the language's behavioral model.

Third Subgoal: the development of a runtime platform that supports the dynamic synthesis of coordinated user-centric communication services. A functional prototype is developed that demonstrates the feasibility of the presented modeling abstraction. Comparative studies are also performed using the prototype to provide arguments for the claimed benefits of the approach.

Contribution: the extension of the current CVM prototype to provide functionalities for service coordination. To evaluate the cost of developing the WF-CML infrastructure, including supporting tools and execution environments, we measured both static and dynamic properties of the resulting system characteristics. Static metrics include the static measurement of the prototype in terms of various properties that reflects the size of the system, including number of lines of code, number of packages and classes and methods. Performance metrics include the measurement of various dynamic system properties that reflect the runtime performance of system, including memory usage, CPU time and number of threads.

1.3 Outline of Work

The outline of this dissertation is structured as follows:

Chapter 2, the literature review, explains and surveys background techniques that lead to the proposed problem and existing approaches in user-centric communication initiatives, workflow modeling and the combination of the two.

Chapter 3, presents the syntactic design of WF-CML, including domain analysis, the abstract syntax and static semantics of WF-CML models. It also presents the application of WF-CML to model a real-world healthcare scenario.

Chapter 4, states the semantic specifications of WF-CML required for dynamic synthesis, including the use of a labeled transition system for formalizing WF-CML semantics, synthesis algorithms that realized the semantics, and the evaluation of the semantics using Kermeta as a metamodeling tool for model simulation.

Chapter 5, describes the implementation of the CVM prototype and a comparative study to validate the claimed benefits of WF-CML. The experimental studies we conducted showed that using WF-CML lead to reduced development and runtime effort and improved development productivity, comparing with general purpose workflow languages.

Chapter 6, presents a summary of this dissertation in terms of an overview of the contribution and future directions of this research.

CHAPTER 2

LITERATURE REVIEW

This chapter presents background knowledge of this research and the related work. Background on user-centric communication services, model-driven software development (MDSD) and workflow techniques are presented. In addition, we also discuss related work on other user-centric communication initiatives, workflow modeling techniques and approaches for combining workflow and multimedia collaboration.

2.1 Background

We start with introducing the concept of user-centric communication (UCC) and the Communication Virtual Machine (CVM) technology to support model creation and realization of user-centric communication services. We also present fundamentals on model driven software development and workflow modeling techniques.

2.1.1 User-Centric Communication

Advances in communication devices and technologies (e.g. iPhone [2], iPad[3] and Android [29]) are dramatically expanding our communication capabilities and enabling a wide range of multimedia communication applications. Examples range from general-purpose applications that support voice calls, video conferencing, and instant messaging using a one-size-fits-all approach, to specialized applications in disaster management, distant learning and telemedicine. It is likely that the pace of innovation of new communication applications will accelerate even further. However, many of these applications have been conceived, designed and custom-built using a

silo approach with little connection to each other, resulting in fragmented and incompatible technologies [46]. The growing heterogeneities of network infrastructure and communication platforms make it both time-consuming and error-prone to develop new communications applications entirely from scratch. Much of the cost and effort stems from the fact that the heterogeneities and complexities of network-level communication control and media delivery are tightly bound with the diversity of application-dependent collaboration logic. This tight coupling also results in applications that are incapable of responding to changing user needs and the dynamics of underlying networks or devices.

To effectively decouple the development of application logic from the network infrastructure, different initiatives have developed open APIs or application frameworks that allow for rapid creation and deployment of converged communication services. These approaches (e.g.,OSA/Parlay[35], JAIN-SLEE[62], and JAIN-SIP[68]) reduced the complexity of service creation by exposing a single point of interfacing with multiple protocols and resources. They simplified service creation by reducing the amount of telecommunication knowledge required for building communication solutions. However, the creation of communication applications are still limited to IT developers who have to use a general programing language such as Java in their respective IDEs and invoke such API interfaces to create a functional composition or program client-side applications (with potentially sophisticated collaborative logic). The programing-level nature of these APIs or frameworks makes them complex to use, and less usable than the user-level sessions of CVM, see Figure 2.1.

There is a strong demand for an easy and flexible way of building communication applications that are driven by end-users and that support the dynamic nature of communication-based collaboration. The user-centric approach of communication service creation has emerged to address this need. We use the term user-centric

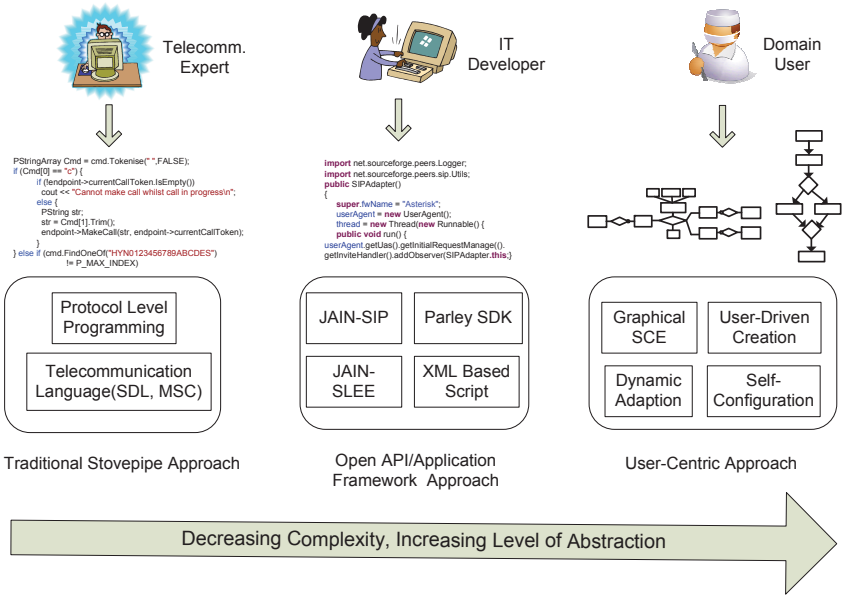

Figure 2.1: Paradigm Shift in Developing Communication Services/Applications

communication to refer to communication applications that are driven by end-users and mask device and network complexity while preserving the diversity and power of advanced communication tools. The trend towards service convergence and user-centricity in communication service development is summarized in Figure 2.1, which illustrates this paradigm shift in creating and realizing communication-intensive services or applications. This evolution is also accompanied by decreasing complexity and increasing level of abstraction of the development process.

2.1.2 Model Driven Software Development

Model-Driven Software Development (MDSD) is typically used to describe software development approaches in which abstract models of software systems are created and systematically transformed to concrete implementations. Model-Driven Software Development (MDSD)[61] provides us with a more systematic support for realizing user-centric communication by letting end-users play a greater role in customizing communication services. It is expected that successful MDSD practices will help to

cope with the ever-present problem of growing software complexity by raising the level of abstraction that software is conceived.

A MDSD process starts with the specification of software solutions using a formal modeling language (e.g. a UML Profile [32], or a domain specific language [27] that captures problem-level abstractions). Then, model transformation techniques are used for bringing problem-level specifications down to low-level languages that are directly executable. During this process, other model management issues need to be addressed as well, including maintaining model consistency, maintaining traceability links among model elements, and using runtime models to observe system behaviors. These are presented as major challenges to fully achieve the MDSD vision in [27]. However, despite the wicked nature of MDSD, various MDSD initiatives have been around and providing useful experiences in this field.

Domain Specific Modeling (DSM): In achieving the MDSD vision, DSM is an enabling field. DSM is concerned with providing support for creating and using problem-level abstractions in modeling languages in a way that is easy to use and facilitates machine automation. Arie et al. [73] defines a domain-specific language as:

A domain-specific language (DSL) is a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain.

In this dissertation, we do not distinguish DSL and domain specific modeling language(DSML), since programs and executable specifications are essentially models. A DSL consists of constructs that capture phenomena in the domain it describes. The final software products can then be generated from these high-level specifications. This automation is possible because the modeling language and generator only need to fit the requirements of one domain.

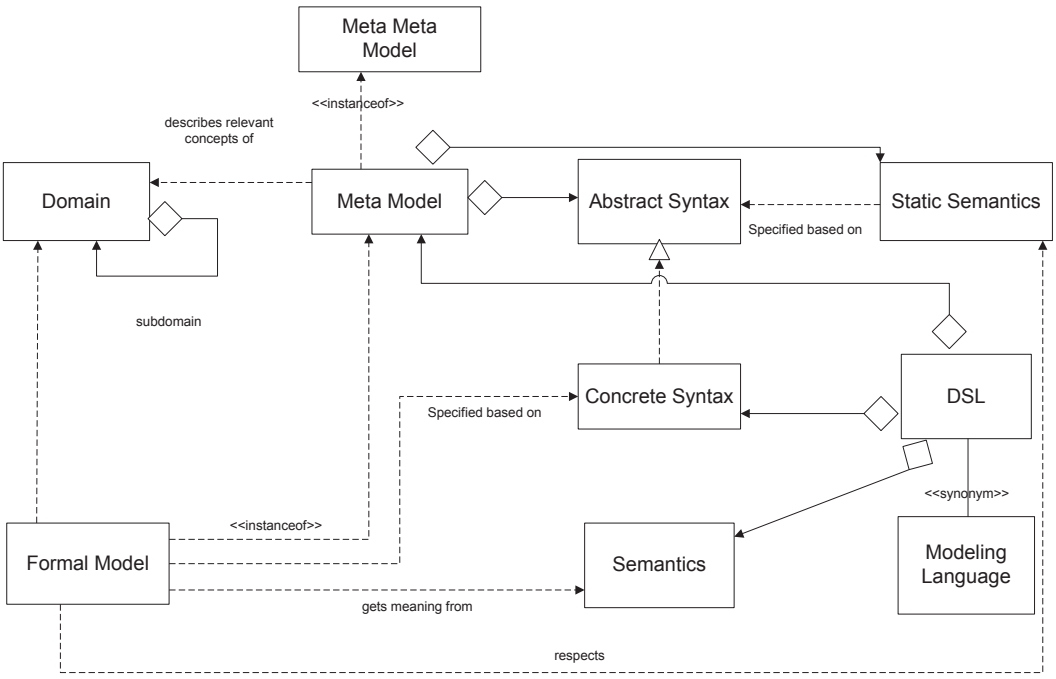

Figure 2.2: Modeling and DSLs: A Roadmap [27]

The development of a DSL typically involves four phases : analysis, design, implementation, and use. The Analysis phase is about the identification of the problem domain and gathering all relevant domain knowledge used as input for the design of the DSL. The design phase produces a specification of the DSL that concisely describes applications in the domain. The implementation phase deals with the construction of libraries that implement the semantics notions and the design and implementation of a compiler or interpreter that translates DSL programs to sequences of library calls. And finally, the Use phase is about writing programs (models) using the DSL for creating applications in the domain.

Volter et al. [74] presented a DSL design roadmap that connects different concepts in a DSL design process, and reflect the set of steps we go through in achieving this work. We show this roadmap in Figure 2.2 [27]. A DSL design always starts from

a domain. The domain describes a bounded field of interest or knowledge. The result of domain analysis is used in the development of several syntactic artifacts including Abstract Syntax, Static Semantics, and Concrete Syntax. Static Semantics are specified based on Abstract Syntax and they are jointly called Meta-Model, which is an instance of a Meta-Meta Model. Then, Semantics are attached to the DSL which provides meaning to a Formal Model specified based on the Concrete Syntax. DSL is a synonym for Modeling Languages. For the rest of this dissertation, we will be using the concepts in this roadmap for the design and development of WF-CML.

During the DSL development process, various language engineering tools could be used to support the development activities [10, 19, 69]. Tool-based language development allows an early detection of problems with the language definition and increases the reliability of the language and its supporting tools. Kermeta [69] is an executable metamodeling language as well as a language development framework for language engineering activities. Built on the Eclipse framework [67], the Kermeta framework provides a powerful metaprogramming environment for metamodel engineering activities including the specification of abstract syntax, static semantics and operational semantics, as well as the rapid prototyping of DSML specific tools. Using Kermeta helps to validate language design decisions early in the development process long before implementation occurs.

Formal semantics for DSMLs: Formal semantics specifications for DSMLs, especially in the MDSD context, is essential for automatic manipulation of models by the underlying platform. Various approaches for specifying formal semantics for DSMLs have been investigated, such as semantics specifications through transformational techniques(translational approach), and semantics descriptions through interpretive techniques(operational approach). In programing language semantics research, a translational approach allows the specification of semantics of a high-level programing

language by mapping to a low-level target language via program transformation. The execution of the target language on a machine captures the semantics of the high-level language. An operational approach, on the other hand, uses an abstract machine, or interpreter, to model how the state of the machine is altered when an program is executed. It could loosely be associated with the operational semantics of programing languages. We explain in more details on how translational and operational semantics could be used for specifying the dynamic behaviors of DSMLs.

Using a translational approach for specifying the semantics of DSMLs usually involve model-to-model transformations that define the semantics of a language model by translating into a target modeling language. The resulting model is usually an executable model, e.g. a program in a general purpose programming language. For instance, the Model Driven Architecture [34], the OMG's standard for MDE, suggests the transformation of platform independent models into platform specific models. By providing standard means for these transformations, QVT forms the center of this approach. Since DSMLs rely on a semantic domain to formalize the semantics specification, the choice of the semantic domain becomes an important issue. Usually, the target domain should be a simpler language whose execution rules are well-established, and facilitate the definition of the "behavior" of each abstract syntax element in the DSML through a suitable mapping on the semantic domain. The role played by graph transformation depends on the chosen formal method. If it is a textual one, the productions of the grammar that defines the abstract syntax can be augmented with textual annotations to build the semantic representation. For example, high-level timed Petri nets are used as semantic domain and the rules are pairs of graph grammar productions: The first production defines how to modify the abstract syntax representation, while the second production states the corresponding changes on the functionally equivalent Petri nets.

The operational semantics of a language, on the other hand, describes the meaning of a language instance as a sequence of computational steps. It originates from the idea of structural operational semantics from grammar-based language engineering practices. Plotkin pioneered the work on structural operational semantics [54](SOS), in which formal semantics of programing languages are given by transition systems. Generally, a transition system $< C, T >$ forms the mathematical foundation, where C is a set of configurations and T is a transition relation. Inference rules are a common way to define the valid transitions in the system inductively. The SOS of a language defines an interpreter for this language working on its abstract syntax. This can be instrumented as a reference to test implementations of compilers and interpreters. Recently, the idea of SOS has been adopted to semantics specifications of DSMLs for model-driven language engineering practices, in which configurations are defined based on meta-models, and transition relations specified based on changes in the metamodel configurations using standard notations such as QVT Relations [33].

The semantics choice of whether to use a translational or an interpretive style for specifying DSML semantics depends on many factors, such as the characteristics of the domain, the purpose of the semantics specification (e.g., validating the prototype, implementations of programing languages, or analyzing more advanced implementations of programing languages). But in general, using translational semantics in a domain-specific context tend to bring many challenging issues such as (1)domain experts find it hard to understand the target language given its low level of details; (2)complicated translations hide the underlying language semantics in the details of the target language;(3)it is a sophisticated task to map error messages and debugging information back into the source domain. Thus, translational semantics do not provide an appropriate level of abstraction for prototyping DSMLs purpose.

Therefore, in many cases, an operational approach is more suited to testing, validation than the translational approach. We stay in the expert domain to describe the operational semantics of the language in terms of its structure (usually syntactic structure). The domain experts understand these structures and should be able to explain the effects of execution. Furthermore, the SOS description is instantly executable in the expert domain. This enables us to test and validate the specified behavior together with the domain experts. Thus, an operational approach is well suited for prototyping.

2.1.3 Modeling User-Centric Communication Services

One of the initiatives for realizing the vision of user-centric communication services is the Communication Virtual Machine (CVM) [17] technology. It consists of a declarative Communication Modeling Language (CML) for specifying users' communication needs, as well as a model driven runtime environment that supports the modeling and realization of user-level communication services using a model driven approach. We limit the scope of the term communication in this paper to denote the exchange of electronic media of any format (e.g., file, video, voice) between a set of participants (humans or agents) over a network (typically IP). Figure 2.3 shows the layered architecture of the CVM. The realization of communication service is divided into four major levels of abstraction, which correspond to the four key components of CVM:

- 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;

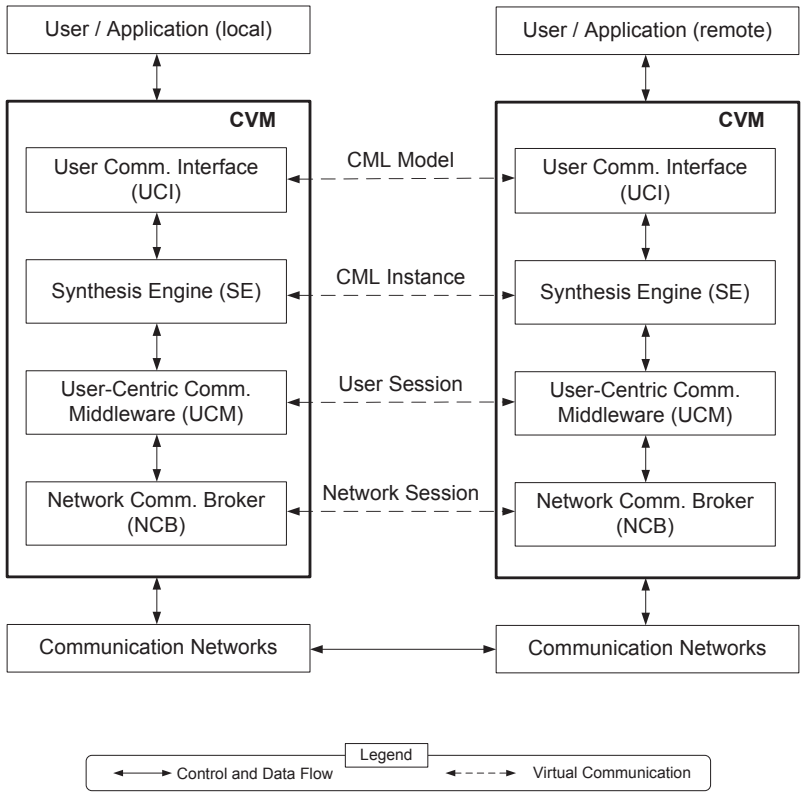

Figure 2.3: Layered architecture of the Communication Virtual Machine.

- Synthesis Engine (SE), generates an executable script (communication control script) from a CML model and negotiates the model with other participants in the communication;
- User-centric Communication Middleware (UCM), executes the communication control script to manage and coordinate the delivery of communication services to users, independent of the underlying network configuration;
- 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 Communication Modeling Language (CML) was developed by Clarke et al. in [11] for modeling user-centric communication requirements. CML was designed to

be simple and intuitive, be independent of the network and device characteristics and be expressive enough to model a majority of user communication models. There are currently two equivalent variants of CML: the XML-based (X-CML) and the graphical (G-CML). The primitive communication operations that can be modeled by CML include: (1) connection establishment, (2) data (primitive and user-defined) transfer, (3) addition/removal of participants to/from a communication, (4) dynamic creation of structured data, and (5) specification of properties associated with a particular data transfer. Figure 2.4 shows the abstract syntax of CML using metamodeling notations. The class diagram shows the various CML constructs that are used in modeling user-centric communication. We also use OCL to specify the static semantics rules for CML, and refer the readers to the project's website[1] for more details.

Two categories of communication models can be described using CML, communication schemas and communication instances. The relation between a schema and an instance is similar to the relation between a class and an object in programming languages. An instance captures all information in a communication at a particular point in time and can be directly executed. On the other hand, a schema describes the possible communication configurations of a conforming instance. To give a look and feel of CML and how to use it to model communication services, we present a CML model for realizing a heathcare scenario in Figure 2.5. As Figure 2.5 shows, CML models a two-way conferencing supporting the transfer of certain types of media (LiveAV)and structured data (Generic Form) in a declarative manner. Note that the intuitive nature of CML is compliant with the concept of user-centric communication.

In the context of Service-Oriented Architecture(SOA), CVM serves as a client-side architecture, as opposed to server-side frameworks such as OSA/Parlay [35] and JAIN-SIP [68]. The server-side architecture has different concerns than the client-

[1] http://www.cis.fiu.edu/cml/

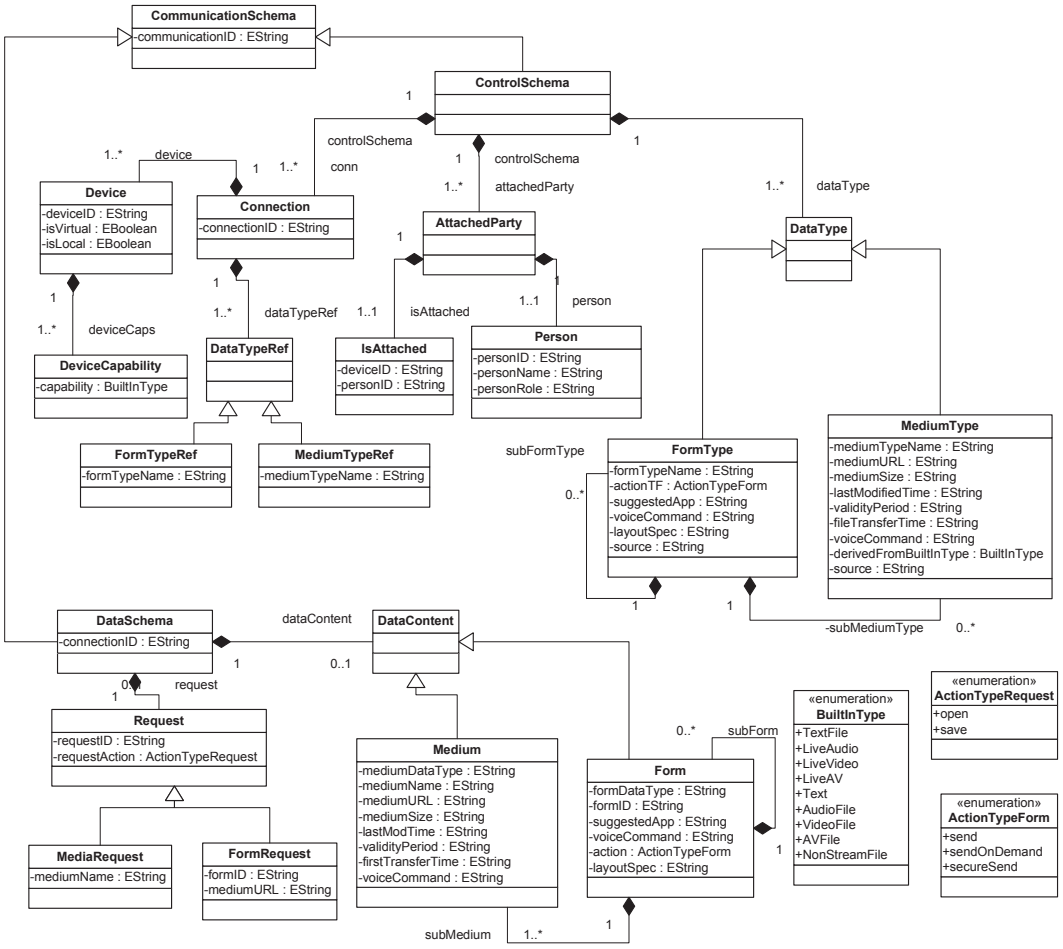

Figure 2.4: Abstract Syntax of X-CML.

side architecture, which is the focus of CVM. As end-hosts are capable of handling sophisticated collaborative logic in IP networks, client-side architecture is getting as important and complicated as server-side architecture in building next-generation multimedia communication applications. In addition, client-side architecture is becoming increasingly pertinent as more and more communication applications are deployed and executed on mobile devices. A big issue for developing communication software on mobile devices is the variety of platforms. For example, to develop communication applications on smart phones, two major platforms are iOS and Android. The CVM architecture, being inherently platform-independent, could be hosted on various mobile platforms.

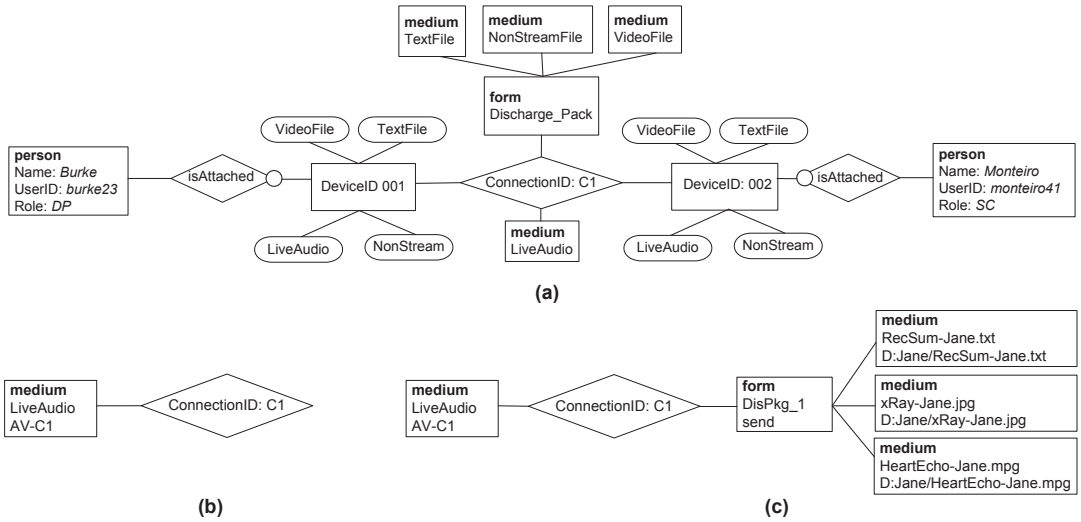

Figure 2.5: G-CML representation for: (a) the control instance for 2-way call between Dr. Burke and Dr. Monteiro, (b) data instance to enable `LiveAudio`, (c) data instance to send form `DisPkg_1`.

2.1.4 Workflow Fundamentals

Workflow is concerned with the automation of procedures where documents, information or tasks are passed between participants according to a defined set of rules to achieve, or contribute to, an overall business goal [12]. Workflow is essential to automate business processes as it coordinates the work of different participants without human intervention. In general, different dimensions of workflow could be specified, including the process dimension (control flow of activities), organization dimension ("who" should do "what") and functionality dimension (functional decomposition of activities). The process dimension of a workflow model defines a template process model which depicts a formalized sequence of separate activities. Each activity is viewed as a unit of work that is performed in a non-zero span of time by an actor. The Workflow Management Coalition (WFMC) [12] defines a process activity as "a logical step or description of a piece of work that contributes towards the accomplishment of a process". Actors are either humans or external systems. They are

grouped into roles according to their responsibility, skills, and authority for humans, or computing capabilities for systems. Assigning roles to activities belong to the organization dimension of workflow modeling.

WFMC defines four possible ordering relationships, or routing rules between activities: sequence, choice, parallelism and iteration.

- Sequence: Tasks are executed sequentially if the execution of one task is followed by the next task.
- Parallelism: Tasks are executed at the same time or in any order. To model parallel routing, AND-Split and AND-Join are used to split into multiple flows and to synchronize multiple flows.
- Choice: Tasks are executed according to certain conditions. To model a choice between multiple alternatives, the Or-Split and Or-Join are used to conditionally branch into alternative flows and re-converge them into one activity.
- Iteration: A workflow activity cycle involving the repetitive execution of one (or more) workflow activity(s) until a condition is met.

Depending on the characteristics of a given workflow modeling language, workflow models might look different and have semantic variations, but the underlying approach for workflow modeling in the process dimension remains: designing a set of activities (tasks) as well as their ordering relationships or routing rules.

Workflow models are interpreted by Workflow Management Systems according to the specified semantics of the model. An executable workflow model requires its semantics be defined in an unambiguous manner. In the literature, various formalisms have been used to design workflow models with precise semantics, including abstract

state machines [5], labeled transition systems [25], and Petri nets [71]. For example, Eshuis in [25] viewed workflow systems as labeled transition systems that react to input events from the environment and respond accordingly by transition relations. Activities are viewed as action states, and wait states are also used to model the local state that is waiting for some external event or temporal event. When an action state is terminated (indicated by occurrence of the activity termination event) or a wait state is completed (indicated by the occurrence of the waiting-upon event), relevant edges are enabled and transitions made according the even-condition-action (ECA) rules of the workflow specification.

Petri nets are also widely investigated for workflow modeling due to its formal semantics and support for reasoning of system properties. Aalst et al. [71] described the use of Petri nets in the context of workflow management. Petri nets are an established tool for modeling and analyzing processes. On the one hand, Petri nets can be used as a design language for the specification of complex workflows. On the other hand, the theory of Petri net provides for powerful analysis techniques which can be used to verify the correctness of workflow procedures. In a specialized Petri net known as the WorkFlow Net (WF-net), workflow concepts are described using Petri net constructs: activities are modeled by transitions, conditions are modeled by places, and cases are modeled by tokens, and other routing constructs modeled by customized places. The token-game semantics of Petri nets yield executable workflow specifications that are readily for processing by the underlying workflow management systems.

Note that the distinction between workflow processes and communication processes should be made. Communication processes focus on multimedia connectivity and the sharing of information rather than the definition of processes. For communication processes, it is not necessary to explicitly define the workflow processes since

it main involves unstructured group activities with no particular order of the collaboration. On the other hand, a workflow process supports a formalized sequence of separate activities to coordinate a work process. There has been some work that investigate the integration of the two different approaches to enables a continuous stream of tasks and activities in which fast, informal, ad-hoc actions can be taken through conferences within the usual formal workflow[78]. The proposed work in this dissertation also brings the advantages of workflow processes and collaborative communication processes in modeling and realizing user-centric communication services.

2.2 Related Work

We investigated three classes of related work. First, existing user centric communication initiatives are presented, including the Alcatel' s user-centric interactions approach, next-generation network approach, and the CVM approach. Then, several workflow modeling languages are surveyed and compared. Finally, we present some other approaches of combining multimedia collaboration and workflow management.

2.2.1 User-Centric Communication Initiatives

Lasserre et al. [48] proposed a user-centric interaction approach that is used by Alcatel to provide a simple user-centric experience that masks device and application complexity while preserving the diversity and power of advanced communication tools. Alcatel's approach is designed to enable enterprises to obtain the maximum benefit from communication technology by unifying the interactions between the three key elements of any business: people, process and technology. To implement and benefit from user centric interaction, a three-step plan needs to be taken: (1) understanding the users in terms of their requirements, usage patterns and reasons for their frustra-

tion; (2) revisiting business processes to find out communication-intensive processes and then integrate communication capabilities into these processes; and (3) translate the people and business needs into technology and solution requirements. Alcatel has developed various communication solutions to integrate telephony voice services and business processes, including "IP Communication Server", "an IP Unified Communication Application Suite","Communication Web Service". It is expected that user-centric communication could potentially accelerate business processes sens06.

Industrial user-centric communication initiatives have been described in different contexts such as: providing various customized communication services like teleconferencing and data services to businesses in specific domains [48, 70], modeling of communication environments [4], content applications that interact with communication sessions [18], and aligning user-centric communication with the unified communication paradigm. Unified communication has a similar objective to user-centric communication i.e., simplifying end-user experiences [40]. It aims to provide a way to integrate audio, video and data technologies on IP networks and allow switching among them while masking the complexities of heterogeneous back-end systems. These technologies, however, usually cannot accommodate unanticipated communication needs without first incurring a lengthy development cycle, resulting in a high cost to modify the functionality and interfaces required. Also in these approaches, user-centricity focuses on enhancing end-user experiences by providing rich yet easy-to-use communication applications. In our approach end-users drive the specification of the communication applications using an intuitive notation that provides them full control over the application functionality.

Recently, along with the emergence and popularity of Next Generation Networks (NGN), various convergent environments have appeared that provide user-centric service creation and delivery facilities. These include OPUCE (Open Platform for

User-centric service Creation and Execution) [84], SPICE (Service Platform for Innovative Communication Environment) [63], among others [50, 58]. These environments support the combination of Internet IT services with communication services such as presence, voice calls and audio/video conferencing. They broadened the range of service creators and show directions towards user-centricity. While these approaches enable non-technically skilled users to create, manage and share their own convergent services, many of them lack the ability to coordinate individual communication services using the appropriate level of abstraction. For instance, the users are still required to choose or select the "base" services that they want to reuse, and build service mashups by composing two or more "base"services. Also, since mashup execution is driven by event triggers, users have to specify the service composition logic through a directed graph or flowchart like diagrams.

We argue that the CVM approach hides the details of service logic by presenting an abstract and declarative interface that does not require imperative encoding. In CVM, end-users are not aware of the specific set of services and resources available and only focused on their high-level application needs. Service composition is made very natural by updating the CML model to incorporate additional service specifications (such as transferring new media types and domain-specific forms). This dissertation focuses on the logic for orchestrating different services via workflow-like constructs. In addition, there is a trade-off between domain-specificity and service generality in CVM, so that it can adopt a lightweight execution engine instead of a heavyweight open service creation and delivery platform.

2.2.2 Workflow Modeling Languages

There exist many workflow modeling languages with various specializations on different domains(e.g.,[39],[79],[21],[36] [28] and [23]). Some languages are designed for the

Modeling Technique	*Domain Scope*	*Modeling Target*	*Executability*	*Base Model*
UML Activity Diagrams	General process modeling	Describe flow of control of a target system	no	null
BPML	Business process modeling	Describe business processes by a common notation	no no	null null
BPEL	Business process modeling	Specifying business process behavior based on interaction between web services	yes	web services
YAWL	Business process modeling	Specifying workflow systems, integration with organizational resources, web service integration modeling	yes	web services
WebWorkflow	Interactive web application dev.	Web applications that contain the coordination of diff. activities performed by parties	yes	WebDSL
WF-CML	user-centric communication	Dependencies between comm. activities at the app. level	yes	CML model

Table 2.1: Comparison Between Different Workflow Techniques

specification of a wide variety of workflows or business processes ranging from office task automation to patient discharge in a hospital setting; while other languages are more focused on a specific domain such as web application development. In this subsection, we will survey four general purpose workflow modeling languages including BPEL [39], BPMN [79], UML Activity Diagrams [21] and YAWL [72], as well as several domain-specific workflow modeling languages, including WebWorkflow [36]. General purpose workflow modeling languages such as those investigated in this work are designed to cover a broad spectrum of workflow design notations. The generality results in a lack of support for automation and integration with other languages necessary for generating full applications, while the domain-specific ones provide a customized workflow solution.

BPEL4WS (Business Process Execution Language for Web Services) [39] defines a notation for specifying and implementing business process behaviors based on Web Services. It is layered on top of web service models defined by WSDL and specifies the interactions between these services in a platform-independent manner. The language defines a set of basic activities: those that describe elemental steps of a process behavior such as the invocation of a web service, as well as structured activities: those that encode control-flow logic, and therefore can contain other basic and/or

structured activities recursively. The structural activities include control flow constructs expressing sequence, branching, parallelism, synchronization, etc. There has been various approaches to express the BPEL semantics in a formal language. Most of that work is focused on proving certain properties; an approach where a formal system, in this case a process description, is transformed into a mathematical model, such as Petri Net [53], Abstract State Machine[5], Pi Calculus [55]. A property of the mathematical model, for example the reachability of activities may then be proven to be a property of the original system. Due to the lack of complete formal semantics in the original specification, BPEL is often used in conjunction with Java to fill in the "missing" semantics. As a result, BPEL is often tied to proprietary implementations of workflow or integration broker engines.

BPMN (Business Process Modeling Notation) [79] is standard for business process modeling, and provides a common graphical notation for specifying business processes in a Business Process Diagram (BPD). The objective of BPMN is to enable better mutual understanding between different stakeholders from business analysts to technical developers. The modeling in BPMN is made by simple diagrams with a small set of graphical elements (three flow objects and two connection ojects, swimlanes and artifacts). It is designed to be easy to use for business users as well as developers to understand the flow and the process. While the main intent of BPMN is to standardize the business process design notations, its lack of semantics leads to ambiguity and confusion in sharing BPMN models. Graphical models specified in BPMN need to be mapped to executable environments as BPEL, and has been implemented in a number of open source tools [30]. However, the development of these tools has exposed fundamental differences between BPMN and BPEL, which make it very difficult, and in some cases impossible, to generate human-readable BPEL code from BPMN models.

UML Activity Diagram [21]: As a workflow specification language, the expressiveness and adequacy of UML activity diagrams for workflow specification, has been systematically evaluated in [21]. It is shown that UML activity diagrams, as specified by the OMG specifications [32], are expressive enough to model the majority of constructs needed in workflow models. Unfortunately, the syntax and semantics of activity diagram defined by OMG [32] are only partially formalized and leave room for semantic ambiguities. This prevents the adoption of activity diagrams in the context of model driven software development. Also, since UML activity diagrams focus on the control aspects of workflow systems, it can not model a complete workflow system alone, and needs to be integrated with other languages or other system code to produce executables.

Yet Another Workflow Language (YAWL) [72]: YAWL is proposed as a result of a rigorous analysis of existing workflow management systems and workflow languages. Aals et al. [72] find that none of the available workflow management systems could provide suitability for all the workflow patterns and that the expressive power of contemporary workflow management systems leaves much to be desired. They decide to create a new workflow language based on Petri Net. As a complete workflow language, YAWL is intended to capture most of the workflow patterns in contemporary workflow management systems. YAWL extends the class of Petri Net with additional features (e.g., composite tasks, OR-joins, removal of tokens and directly connected transitions), to facilitate the modeling of complex workflows with multiple instances. The operational semantics of YAWL are described using transition systems inspired by token concepts in colored petri nets. Workflow states are defined as bags of tokens each token represented by its location and identifier, and state transitions rules are defined using special "binding relations". Finally, the language is supported by an extensible software system including an execution engine, a graphical editor, and a

worklist handler. The YAWL system is extensible by allowing external applications to interconnect with the workflow engine using a service-oriented approach.

The more expressive a modeling language is, the more difficult it might be to formally reason about the language and define executable semantics for automation. For instance, the one-size fits all approach of the above mentioned BPMN and UML activity diagram makes it difficult to automate the execution of these models without precisely defined formal semantics. This in turn motivates researchers in designing domain specific workflow modeling languages that facilitate rapid realization of workflow applications by targeting at a specific application domain.

There has been plethora of work on defining workflow modeling languages for specific domains. Hemel in [36] proposes WebWorkflow, which extends WebDSL, a domain specific language for web application development, with workflow abstractions. The extension is realized by means of model transformations in which the target model is a core WebDSL. Since WebWorkflow only focus on interactive web application development, domain concepts like high level data model, user interface, procedural events, and access control abstractions could be first-class citizens in the modeling phase. Webworkflow allows the generation of a complete application that can be customized at the modeling level to better suit application-specific needs.

Freutenstein et al. in [28] proposed the model-driven construction of workflow based web applications with domain specific languages. In this approach, a core DSL is used for workflow modeling which supports various modeling notations like BPMN or Petri nets, and a set of DSLs are outlined for designing workflow activities like dialog construction, data presentation and web service communication. Rich workflow-based web applications can be built by modeling workflow and activities and pass them to an associated technical platform. Again, dedicated modeling constructs necessary for generating running workflow-based web applications are included in the

DSL, and activity building blocks are used for the realization of different types of activities. The ongoing research work in domain-specific workflow modeling provide insights for our work of integrating workflow into user-centric communication.

As a summary of our investigation, we presented the comparison between different workflow modeling languages in Table 2.1. We distinguished between Domain Scope, Modeling Target, Executability and Base Model of these languages. We could see that by restricting the domain scope and modeling target of these languages, the workflow abstractions have direct executability as opposed to general purpose workflow modeling.

2.2.3 Combining Workflow with Multimedia Communication

The integration of workflow concepts and multimedia communication systems is not new. Weber et al. [78] proposed the integration of multimedia collaboration (MMC) tools into WFMS in order to furnish a synchronous collaboration work. To model conferences in a workflow environment, he introduced a new modeling construct "conference activity" to denote activities that take place during a MMC conference. Conferences could either be pre-scheduled, or ad-hoc, depending on if the conference activities could be foreseen at model time. The integration of MMC and WFMS occurs through a separate conference broker, which serves as a mediator between the two systems. The broker takes conference descriptions from the WFMS and pass it to the MMC system. The disadvantage is that users have to model the workflow using a workflow modeling language, and use another language for modeling conference descriptions, making it difficult to use in a real-life scenario. By using a unified modeling approach for bringing together the workflow and the communication requirements, users could more easily specify their communication process needs without switching between tools. Another limitation of this approach is the fixed coupling of a WFMS

with a particular MMC system. Bringing the integration to the modeling level results in more flexibility and platform independence.

In [56], a multimedia workflow-based collaborative engineering environment(CEE) is proposed. The environment is composed by the integration of WFS, MMC systems, and collaborative virtual environment. It is intended to control the execution of engineering projects involving many geographically distributed teams and allow an easy integration of different applications providing the team workers with means of information exchange. However, similar to Weber's approach, the proposed CEE environment only addresses the integration from a system and architectural point of view, without regarding for integration of the different concepts at the modeling level. More research needs to be done to clearly specify the conceptual integration as well as to refine the architectural integration. The lack of appropriate modeling techniques for integrating workflow with multimedia communication motivates the proposed research problem presented in the next chapter.

CHAPTER 3

WF-CML: WORKFLOW COMMUNICATION MODELING LANGUAGE

In this section, we present the syntactical design of the Workflow Communication Modeling Language (WF-CML). It starts with an overview of the design goals of the language, followed by a detailed domain analysis of user-centric communication services, including the extraction of communication-intensive healthcare scenarios, and the application of Feature Oriented Domain Analysis (FODA) methodology. Then, we introduce the syntactical design of the WF-CML language that involves its abstract syntax, static semantics, and concrete syntax (including both a graphical and an XML notation. After that, we show how WF-CML is used to model a patient discharge scenario listed earlier in Section 1.1.

3.1 Identifying Design Goals

We identified the following criteria to guide the development of WF-CML:

- should be simple, yet expressive enough, to model the coordination of UCCSs by domain experts
- should support rapid realization - dynamic synthesis and automatic execution of models
- supports dynamic adaptation of executing models at runtime

The need for seeking simplicity and expressiveness in the language design motives us to raise the level of abstraction at which communication solutions should be specified, and is the main reason that we adopted a domain-specific modeling approach.

The need for being amenable to rapid realization requires the semantics of the language be specified and synthesis techniques be designed that are independent from network and device characteristics. Finally, the need for dynamic adaption of services at runtime requires the semantics be specified in a flexible manner.

CVM is intended to support lightweight and agile communication processes, and thus we chose to extend CML with the necessary constructs rather than integrate a full-blown workflow modeling language. Furthermore, instead of software developers, CVM targets at "domain experts", persons within a communication-intensive application domain (e.g., healthcare) that have some IT knowledge, but are not software engineers or programmers. The "domain-specificity" of our approach results in reduced development effort for creating coordinated communication services.

3.2 Domain analysis of User-Centric Communication

The purpose of domain analysis of user centric communication services is to extract recurring behavioral patterns in communication, which are then used as input to the design of a workflow language targeted for communication orchestration. The output of the domain analysis is a domain model (ontology) describing the domain concepts and their relationships. In this section, we presented three scenarios from the user centric communication domain and applied Feature-Oriented Domain Analysis to extract a domain model (in the form of a feature diagram) from these scenarios.

3.2.1 User-Centric Communication Scenarios

We investigated several communication intensive domains like healthcare, disaster management, and scientific collaboration. For each of these domains, a representative

set of scenarios is chosen. We present three scenarios from the healthcare domain to motivate the need for collaborating user-centric communication services:

Patient Discharge Scenario: (1) On the day of discharge, Dr. Burke (DP) establishes an audio communication with Dr. Monteiro (SC) to discuss the discharge of baby Jane. During the conversation, Dr. Burke sends Jane's discharge package to Dr. Monteiro for validation. The discharge package consists of a summary of the patients condition (text file); x-Ray of the patient's heart (non-stream file); and an echocardiogram (echo) of the patient's heart (video clip). (2) After the package is sent, Dr. Burke contacts Dr. Sanchez (PCP) to join the conversation with Dr. Monteiro to discuss the patient's condition. During the conversation, Dr. Monteiro validates Jane's discharge package and sends it back to Dr. Burke. (3) If the discharge package is received within 24 hours and is validated, Dr. Burke then sends it to Nurse Smith (NP) and Dr. Wang (AP). Otherwise, Dr. Burke sends out an interim discharge note (text file) to the AP. At the same time Dr. Burke continues his conference with Drs. Monteiro and Sanchez.

Medical Emergency Response Scenario: Patient X presents to emergency room of General Hospital in State A. She has been in a serious car accident. The patient is an 89 year old widow who appears very confused. (1) The law enforcement personnel Y investigating the accident contacted her adult daughter to indicate that the patient was driving and there are questions concerning her possible impairment due to medications. Her adult daughter informed the ER staff that her mother has recently undergone treatment at a hospital in a neighboring state B and has a prescription for an antipsychotic drug. (2) The emergency room physician W determines there is a need to obtain information about Patient X's prior diagnosis and treatment during

the previous inpatient stay and requested the required information to the hospital in state B.

Disaster Management Scenario: The policy department responds to a fire on the FIU (Florida International University) campus near the biology labs; on arriving the police establish a communication with the Sweetwater fire department and FIU library using the level 1 emergency communication workflow for a fire on campus. The FIU library is contacted to obtain the blueprints for the Biology labs which are downloaded and shared with the policy and fire department. When the fire squad arrives on the scene, it is established that there is hazardous material in the biology labs ad the FIU police updates the level 1 workflow by integrating the level 3 emergency communication workflow and deploys it. The level 3 communication workflow contacts the Dade County fire chief, Dade County health supervisor and the Dade County hazmat team.

3.2.2 Feature Oriented Domain Analysis

From the identified set of scenarios, we perform domain analysis using the FODA (Feature-Oriented Domain Analysis) [9] methodology to yield a taxonomy model for coordinated user centric communication. The taxonomy model is presented in the form of a feature diagram that captures commonalities (mandatory features) and variabilities (variable features) of the domain. It also represents a hierarchical decomposition of the features and their character, that is, whether they are mandatory, alternative, or optional. More specifically, we extended the feature-oriented domain analysis done by Wang [75] to include workflow related features thus making communication services workflow-aware. Figure 3.1 shows the feature model that represents the domain model for user-centric communication services. The class of user-centric communication services has a hierarchy of features that characterize common and

variable domain properties. The root of the hierarchy, "Communication Service", has "communication flow" as a required feature, which may be a trivial flow consisting of one communication process ("basic communication"), or a set of communication processes coordinated by a control flow. Basic communication involves (1) connection, (2) media (primitive and composite) (3) devices for supporting the connection and media transfer, and (4) user behaviors for setting up the connection or initiating data transfer. Connection includes participants in the connection and connection properties. We show several control constructs for coordinating the behavior of basic communication activities, including (1) Sequencing of communication processes, (2) Branching and merging of communication processes, (3) Repetition of communication activities, and (4) Parallel and synchronization of several communication processes. The service management includes features related to loading, saving and runtime adaption of communication services, and is usually offered by the service platform. The rest of the concepts in the feature diagram are self-explanatory. Note that the identified features are by no means an exhaustive set. We chose essential features in order to build a minimal, intuitive and adequately expressive first version of WF-CML. Advanced features such as requested properties for a connection or a particular data transfer are postponed for later versions of the language. These features include quality of service policies (e.g., if bandwidth is low then do not send video streams), security, access rights policies, and timing commands (e.g., transfer the sensor output every 5 seconds). Note that we follow an inductive approach by incrementally introducing workflow abstractions that allow one to capture a set of common programming patterns for a particular domain. This is in contract with a deductive approach, where an exhaustive domain analysis process needs to occur before any language design activities. The inductive approach enables a quick turn around time for the development of workflow abstractions in the modeling of communication services.

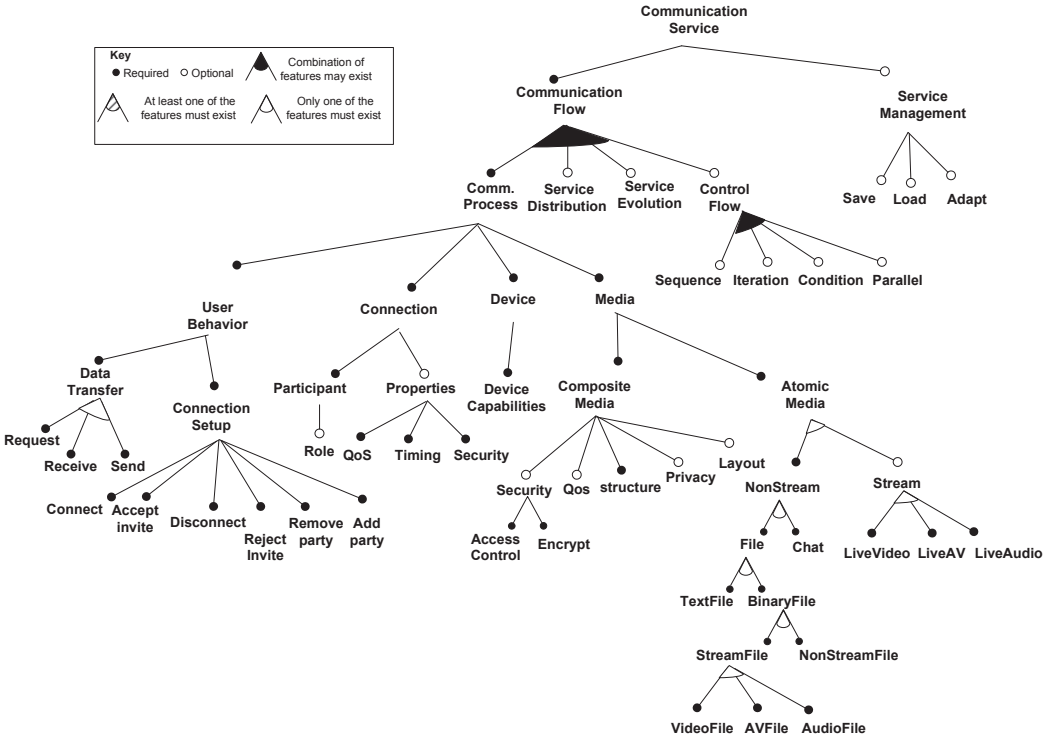

Figure 3.1: Feature Oriented Diagram for the Communication Domain.

Figure 3.1 shows the domain of user-centric communication service consisting of two features: basic communication, coordinated communication and service management. Coordinated Communication includes various control constructs for coordinating the behavior of basic communication activities. The service management includes features related to loading, saving and runtime adaption of communication services, and is usually offered by the service platform. The rest of the concepts in the feature diagram are self-explanatory.

3.3 Abstract Syntax of WF-CML

As indicated in the previous subsection, by investigating domain-specific scenarios in healthcare, disaster management, we have gained preliminary results in terms of the most common patterns and event. These results have helped us to identify the

modeling abstractions for coordinated communication services. Some of the modeling constructs we have identified include: Communication Process, Workflow Event, Decision and Merge. Each Communication Process incorporates a CML model that represents the functionality of the communication service. That is, each Communication Process includes a CML model, the execution of which realizes the services specified. Also, a specific type of Communication Event is attached to the activity as trigger or temporal event, indicating the termination of this activity. Communication events drive the execution of WF-CML models through the initiation, termination and triggering of communication activities.

To perform a classification of the events that could potentially drive the progress of communication workflows, we categorize Workflow Event into four categories: Negotiation Event, Media Transfer Event, Temporal Event and Exception Event. Negotiation Event indicates the result of a connection establishment, and could be either Failed, or Successful, or Updated, while Data Event signifies the reception, or sending of a particular medium, and could be sub-categorized into Form Event and Medium Event. The Temporal Event and Exception reflects a time occurrence and an unexpected failure separately. Note that the Temporal Event are related with a Negotiation Event or a Data Event to denote the start of a time count. Therefore,Temporal Event could be represented by temporal properties integrated with Negotiation Event or Data Event, as shown later.

We defined the meta-model of WF-CML using an abstract syntax, shown in Figure 3.2, and static semantics described in the next subsection. In short, a WF-CML model is a graph (CommWorkFlow) consisting of nodes (WF-Node), edges (WF-Edge), and trigger events (TriggerEvent) as shown in Figure 3.2.

InitialNode and FinalNode signify the beginning and ending of a model representing the coordination of communication processes. CommProcNod (communication

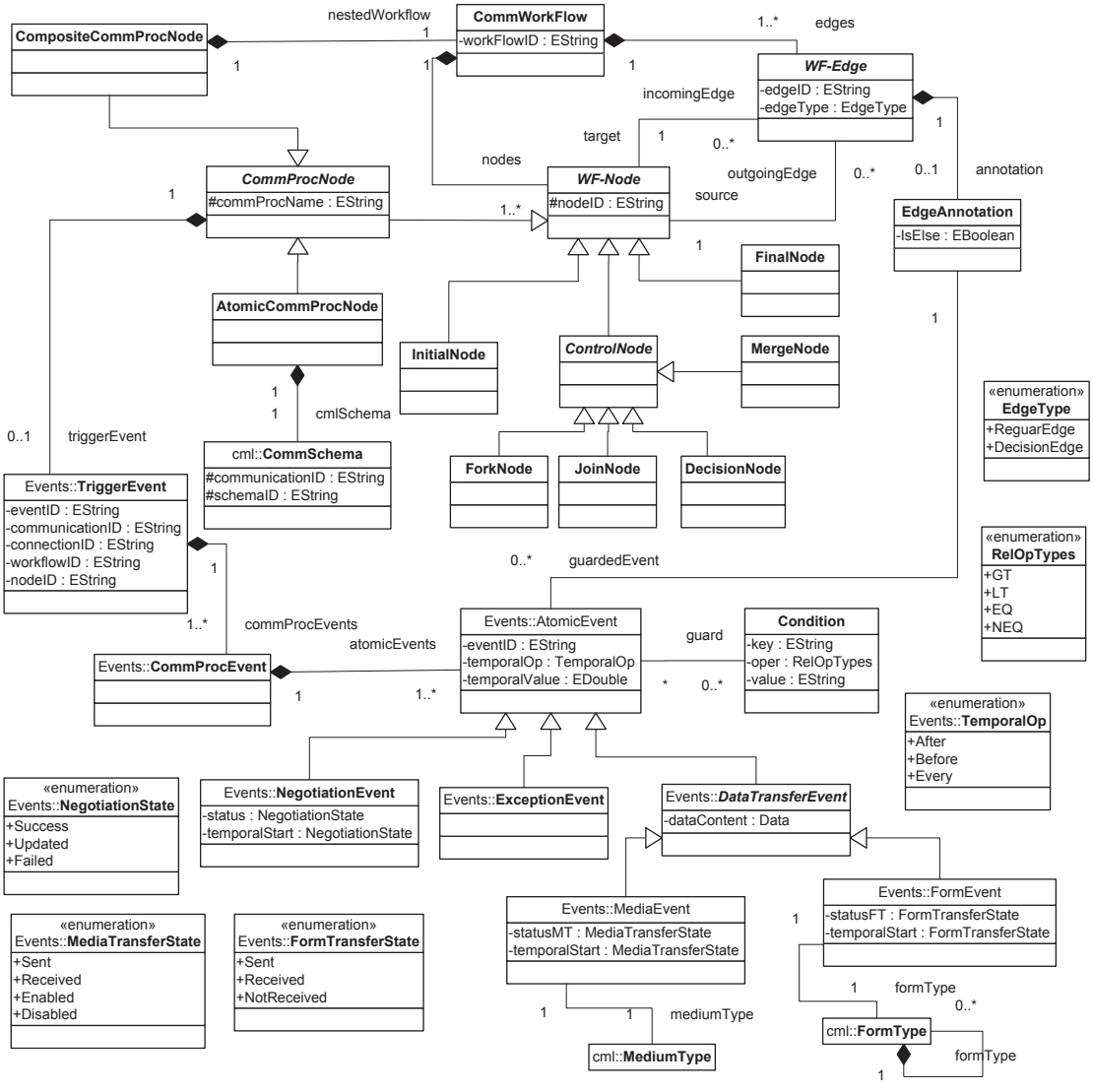

Figure 3.2: Abstract Syntax for WF-CML Using a Class Diagram.

process node) is either an atomic communication model (AtomicCommProcNode) or a nested workflow model (CompositeCommProcNode) and has zero or one trigger events associated with the node. The atomic communication model has a CML model (cml::Comm Schema) and represents a communication service between participants, an example of which is shown in Figure 2.5. The meta-model for CML is also available on the project's website[1]. We will use the term CS process to refer to an atomic communication process, and communication process to refer to either a composite communication process or an atomic communication process from this point on in the paper. DecisionNode, ForkNode, JoinNode and MergeNode express control flow between communication processes. There are two types of edges (decision and regular). A decision edge is annotated with zero or more atomic events. If there is no event annotation on the decision edge it is considered an else edge.

TriggerEvent is composed of one or more communication process events termed as CommProcEvent. AtomicEvent may be either a NegotiationEvent e.g., "negotiation success"; ExceptionEvent e.g., "connection interrupted"; MediaEvent e.g., "file A received"; or a FormEvent e.g., "form DisPkg_1 received". Each atomic event may have a temporal property associated with the event e.g., "DisPkg_1 not received 24 hrs after being sent". To support the definition of trigger events we defined several temporal operators (TemporalOp), negotiation states (NegotiationState), media transfer states (MediaTransferState), and form transfer states (FormTransferState).

We also specify the static semantics for the integrated WF-CML metamodel using OCL, which is discussed in the next subsection.

```
1. Context CommWorkFlow
   inv: (Self.nodes.getCommProcNodes() >0)
   and       (Self.nodes.forAll(node| not node.oclIsTypeOf(DecisionNode) implies
       node.outgoingEdge.forAll(e| e.edgeType== edgeType::regularEdge))
   and  (Self.nodes.select(n|n.oclIsTypeOf(DecisionNode))−>size() ==
       Self.nodes.select(n|n.oclIsTypeOf(MergeNode))−>size()
   and  (Self.nodes.select(n|n.oclIsTypeOf(ForkNode))−>size() ==
       Self.nodes.select(n|n.oclIsTypeOf(JoinNode))−>size()
2. Context InitialNode
   inv: (Self.outgoingEdge−>size()==1)
   and       (Self.incomineEdge−>size()==0)
3. Context FinalNode
   inv: (Self.outgoingEdge−>size()==0)
   and (Self.incomineEdge−>size()==1)
4. Context DecisionNode
    inv: (Self.outgoingEdge−>size() >1)
    and (Self.incomingEdge−>size()==1)
    and (Self.outgoingEdge.forAll(edge|edge.edgeType == edgeType::DecisionEdge)
    and (Self.outgoingEdge.select(edge| edge. annotation.isElse == true)−> size() ==1
    and (Self.incomingEdge.source.triggerEvents.forAll(tr | self.outgoingEdge−> exits
       (edge| tr ==  edge. annotation. guardedEvent))
5. Context DecisionEdge
   inv: not (Self.annotation.oclIsTypeOf(OclVoid))
6. Context RegularEdge
   inv: Self.annotation.oclIsTypeOf(OclVoid)
7. Context MergeNode
   inv: (Self.outgoingEdge−>size() == 1)
   and (Self.incomingEdge−>size() > 1)
8. Context ForkNode
   inv: (Self.outgoingEdge−>size() >1)
   and  (Self.incomingEdge−>size()==1)
9. Context JoinNode
   inv: (Self.outgoingEdge−>size() ==1)
   and (Self.incomingEdge−>size() >1)
   and (Self.incomingEdges.forAll(edge| edge.edgeType == edgeType::RegularEdge)
```

Figure 3.3: Static Semantics for WF-CML

3.4 Static Semantics of WF-CML

Figure 3.3 presents the static semantics for WF-CML specified as OCL invariants. These invariants correspond to structural constraints that are not captured by the abstract syntax. The first rule states that a WF-CML model must have at least one Communication Process node, for any node that is not a Decision node the outgoing edges are Regular edges, the number of Decision nodes is equal to the number of Merge nodes, and the number of Fork nodes is equal to the number of Join nodes. Another constraint specifies that decision nodes should have two or more outgoing edges, each with edge annotations as a guard except one with the reserved word "else" as a guard. We also specify restrictions for the guards on the decision edges.

[1]http://www.cis.fiu.edu/cml/

A restriction for decisions (and any edge in general) in UML Activity Diagrams is that guards are not allowed to use data from the activity context (i.e., only the data from the current token may be used). Similarly in WF-CML, guards are predicates based on the triggering events for the associated communication process node, not from other data sources. Other rules are fairly easy to follow. We have posted a more complete set of semantic rules in Appendix A19.

3.5 Concrete Syntax of WF-CML

In defining the concrete syntax of the WF-CML language, we extended the CML concrete syntax with graphical notations for modeling the coordination of communication processes. Since CML has two forms of concrete syntax: a graphical representation (G-CML) as well as an XML representation (X-CML), the addition of workflow notations will be defined in both graphical and XML formats.

Visually, the concept of communication processes could be viewed as a compartment that includes the configuration of the communication services, and potentially events that will trigger the advancement of the process flow. Therefore, we design the graphical notations for Communication Process as an nesting rectangle. The containment relation between Communication Process and Communication Schema is also represented in spacial containment. Control nodes are assigned different shapes as follows: and-control nodes (Fork and Join) are represented as horizontal bars, or-nodes represented as diamonds. The specification of X-CML formats for WF-CML is more straightforward due to the strict mapping from the abstract syntax of WF-CML to its XML presentation. We integrated different XML tags for the workflow constructs in the XML Schema Definition (XSD) of the Communication Modeling Language to match the identified abstract syntax in Figure 3.2. Abstractly, classes in the WF-

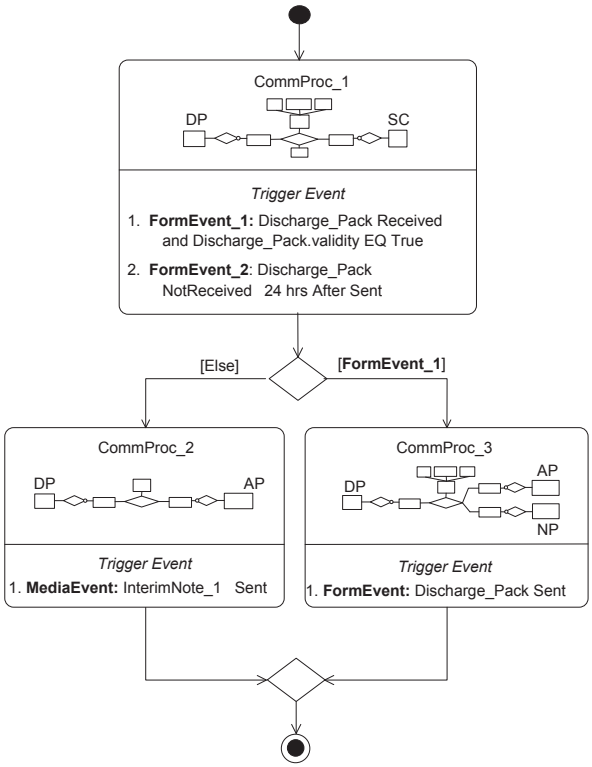

Figure 3.4: WF-CML model for Healthcare Use Case.

CML meta-model that have aggregation or composition relations are mapped to child XML tags inside the parent XML tags. An association relation, on the other hand, is mapped to an additional XML tag to connect the two classes. The XML format for the patient discharge scenario is listed in Appendix A1-A4.

Note that the G-CML representation are intended for user-friendliness purposes, while the X-CML representation are designed for the internal representation and manipulation. They are equivalent in terms of expressive power.

3.6 Modeling with WF-CML

We can use WF-CML to model communication-intensive use cases that coordinate individual communication services. As an example, we show the WF-CML model representing the patient discharge scenario presented in Section 3.2 in Figure 3.4. The

model includes three communication process nodes, each one containing a separate CML model and a trigger event. The CML model in `CommProc_1` specifies the communication between the DP and the SC, which is instantiated when the WF-CML model is executed by Dr. Burke and he loads the contact information of the SC. There are two types defined for this communication, a form type `Discharge_Pack` and a built-in media type `LiveAudio`. The media type and form type are instantiated when Dr. Burke enables the audio stream, Figure 2.5(b), and loads the patient form `DisPkg_1`, Figure 2.5(c), respectively. The trigger event in `CommProc_1` states that this node is exited when a validated patient form of type `Discharge_Pack` is received, in this case `DisPkg_1`; or the patient form is not received 24 hours after being sent. The occurrence of either event will result in the communication flow leaving the node `CommProc_1`. Then, depending on which event triggers the exit, either the `CommProc_2` node or the `CommProc_3` will be activated. Finally, in either cases, a `MediaEvent` (`InterimNote_1_Sent` or `Discharge_Pack_Sent`) will take the flow out the respective node and the flow of communication will terminate afterwards.

3.7 Chapter Summary

In this chapter, we introduced the syntactic design of the Workflow Communication Modeling Language (WF-CML) for coordinating user-centric communication services. It consists of a feature oriented domain analysis, a meta-model and concrete syntax. The syntactic design of WF-CML defines all the first-class modeling constructs of the language, as well as its " look and feel" for the user. An equally important, if not more important task is the semantics specifications of WF-CML. In the next chapter, we introduce the dynamic semantics of WF-CML that provides the basis for dynamic synthesis and rapid realization of coordinated communication services.

CHAPTER 4

SEMANTICS FOR DYNAMIC SYNTHESIS OF WF-CML

In this chapter we describe the dynamic semantics (behavioral model) of WF-CML that enables dynamic synthesis of coordinated user-centric communication applications. We first present an overview of the process to dynamically synthesize WF-CML models, and then formalize the high-level semantics of WF-CML models using labeled transition systems. The semantics specifications are then instantiated by synthesis algorithms and detailed state machines for negotiation and media transfer. To partially validate the semantics, we simulated the WF-CML models using a meta-modeling language development framework that supports the rapid prototyping of domain-specific modeling languages.

4.1 Overview of Dynamic Synthesis For Realizing WF-CML

In this section we provide an overview of the process to dynamically synthesize WF-CML models. We use the term *synthesis* to refer to the automatic derivation of an executable form called *communication control script* from a WF-CML model; and *dynamic* refers to the possibility of runtime updates to an executing WF-CML model. These runtime updates may include adding new media types to an executing communication application, or adding a new participant, which requires renegotiation, or changing the flow of communication services. We first present a high-level flow of the synthesis process of realizing WF-CML models, and then introduce the *hypergraph* concept and explain how it is used during the synthesis process.

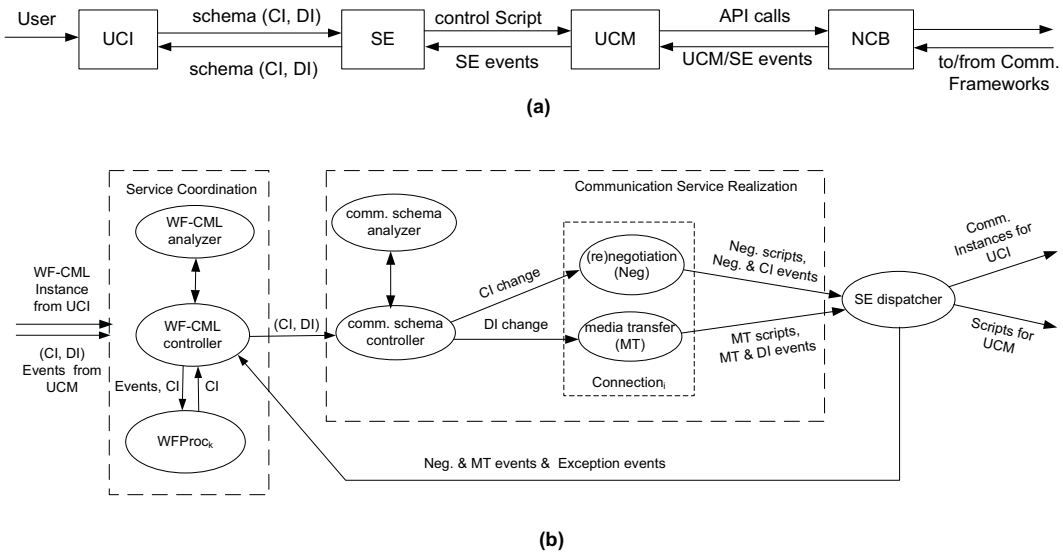

Figure 4.1: (a) Execution of a schema in the CVM. (b) Execution of a schema in the synthesis engine (SE). CI - Control instance; DI - Data exchange instance.

4.1.1 High-Level Synthesis Process

To assist the reader in understanding the process of dynamic synthesis, we show a high-level view of the overall realization process in Figure 4.1(a) and dynamic synthesis in Figure 4.1(b). Recall that realization refers to the dynamic synthesis of control scripts from WF-CML models and subsequent execution of the behavior specified in the models by executing these control scripts. As Figure 4.1(a) illustrates, realization of WF-CML models starts when the user loads and instantiates a WF-CML model in UCI. The realization process could be summarized as the following steps:

1. UCI validates the WF-CML instance with respect to well-formed rules and passes it along to SE

2. SE, upon receiving a WF-CML instance, analyzes and synthesizes it into a control script to be executed by the UCM. It also maintains the runtime state of communication services and service coordination as specified in the model

3. UCM then executes the script and makes API calls to the NCB, which interfaces with the underlying communication frameworks

4. NCB interacts with the communication frameworks and generates UCM or SE events that are handled by their respective CVM layers.

In the above realization process that involves all CVM layers, SE is the layer that specifically handles the dynamic synthesis. Figure 4.1(b) shows a block diagram representing the executing processes in SE. The seven major processes of SE include: *WF-CML Controller*, *WF_Proc*$_i$ (representing the ith process in the WF-CML model), *WF-CML Analyzer*, *Comm. Schema Controller*, *Schema Analyzer*, *Connection*$_j$ (representing the jth connection in the CML model), and *SE Dispatcher* [76]. We describe the functionality of each component as follows:

- *WF-CML Controller*: coordinates the execution of communication services as specified in the WF-CML model by starting, updating, or deleting an executing communication process (represented in the *WF_Proc*$_i$). It interacts directly with the *Comm. Schema Controller* for realizing communication services, and receive negotiation, media transfer and excepetion events from the *SE Dispatcher* to trigger the progression of communication processes.

- *WF_Proc*: represents a specific currently executing WF-CML process. It is coordinated directly by the *WF-CML Controller*. Each *WF_Proc*$_i$ executes independently from each other.

- *WF-CML Analyzer*: analyzes a WF-CML instance and converts it to a flattened data structured (a WF-CML hypergraph, discussed in more details later) that will be executed in *WF_Proc*.

1. *controlScript* ::= *command {command}*
2. *command* := *createConnectionCmd* | *closeConnectionCmd* | *addParticipantCmd* | *removeParticipantCmd* | *sendSchemaCmd* | *enableMediaInitiatorCmd* | *enableMediaReceiverCmd* | *disableMediaInitiatorCmd* | *disableMediaReceiverCmd* | *sendMediaCmd* | *sendFormCmd* | *declineConnectionCmd* | *requestFormCmd* | *requestMediaCmd* | *sendNegTokenCmd* | *requestNegTokenCmd*
3. *createConnectionCmd* ::= **createConnection** connectionID_A
4. *closeConnectionCmd* ::= **closeConnection** connectionID_A
5. *addParticipantCmd* ::= **addParticipant** connectionID_A personID_A {personID_A}
6. *removeParticipantCmd* ::= **removeParticipant** connectionID_A personID_A {personID_A }
7. *sendSchemaCmd* ::= **sendSchema** connectionID_A sender-personID_A receiver-personID_A {receiver-personID_A} schema_A
8. *enableMediaInitiatorCmd* ::= **enableInitiatorMedia** connectionID_A mediaName_A
9. *enableMediaReceiverCmd* ::= **enableReceiverMedia** connectionID_A mediaName_A
10. dis*ableMediaInitiatorCmd* ::= **disableInitiatorMedia** connectionID_A mediaName_A
11. dis*ableMediaReceiverCmd* ::= **disableReceiverMedia** connectionID_A mediaName_A
12. *sendMediaCmd* ::= **sendMedia** connectionID_A mediaName_A mediumURL_A
13. *sendFormCmd* ::= **sendForm** connectionID_A formID_A mediumURL_A {mediumURL_A } action_A
14. *declineConnectionCmd* ::= **declineConnection** sender-personID_A receiver-personID_A {receiver-personID_A}
15. *requestFormCmd* ::= **requestForm** connectionID_A formID_A mediumURL_A {mediumURL_A } requestAction_A
16. *requestMediaCmd* ::= **requestMedia** connectionID_A mediaName_A requestAction_A
17. *sendNegTokenCmd* ::= **sendNegToken** personID_A
18. *requestNegTokenCmd* ::= **requestNegToken** connectionID_A

Figure 4.2: EBNF-like Grammar for Communication Control Script

- *Comm. Schema Controller*: manages the execution of incoming CML instances by interacting with the *Schema Analyzer* and *Connection* process. It also maintains and updates the SE environment accordingly during dynamic synthesis.

- *Schema Analyzer*: performs an analysis of the input instance pair and extracts the changes in the model, which are then fed into the *Connection* process.

- *Connection*: consists of *(re)negotiation* and *media transfer* processes. The negotiation process handles the negotiation logic for a given connection instance, while the media transfer process maintains the state of media transfer within the connection.

- *SE Dispatcher*: coordinates outgoing schemas to be displayed to UCI, control scripts to be passed to UCM, and action requests invoked back to the SE controller for updating the SE environment.

We show the attributed grammar of the control script language using EBNF-like notation in Figure 4.2. The grammar contains productions for all of the script commands used in the control script language. Rule 1 states that a control script consists of one or more script commands, and Rule 2 shows the various script commands. The strings in bold represent the actual command, and the attributes represent the parameters that the command can take. For example, Rule 3 states that *createConnectionCmd* is composed of the string **createConnection** and takes one parameter *connection ID*. The complete metamodel definitions for the control script language can be found on our project website [1].

4.1.2 Using WF-CML Hypergraphs During Synthesis

In this subsection, we present an important data structure used in the dynamic synthesis of WF-CML models, the WF-CML hypergraph. Inspired by the concept of activity hypergraphs as introduced by [25], we used the WF-CML hypergraph concept as a simplified runtime representation of the WF-CML model. Next we present the intuition behind and definition of the WF-CML hypergraph concept, and then introduce a two-step process that is used for realizing WF-CML models based on the WF-CML hypergraph.

Control nodes in WF-CML models like Decision and Merge nodes act like syntactic sugar that denote routing rules without mapping into executable specifications. They could be viewed as transient nodes that leave the system in an unstable state, and process execution will immediately continue from there. Existing techniques for eliminating these nodes using the concept of UML activity hypergraphs and hyperedges have been proposed in [24]. Activity hypergraph is similar to an UML activity digram except that control nodes are removed and replaced by a compound tran-

[1] http://www.cis.fiu.edu/cml/

sition (hyperedge) [24]. Note that hyperedges here are defined as edges that have multiple source nodes and multiple target nodes. The compound transition, essentially a hyperedge, basically is a chain of atomic transitions linked by control nodes. The compound transition could be executable simultaneously as a single transition. Figure 4.3 shows an example of eliminating Decision, Merge Join and Fork nodes by replacing them with compound transitions.

Inspired by the activity hypergraphs, we decided to use the concept of hypergraphs and hyperedges [24] in the manipulation of WF-CML models. The immediate benefit is that they provide for the removal of nodes (Decision, Merge, Fork and Join) that do not map to any real communication processes but are used mainly for modeling routing rules. These control nodes can be replaced with "composite transitions", annotated edges, directly connecting communication process nodes. Note the difference between UML activities and communication processes: an UML activity terminates its execution when it advances to the next node while a CS process continues execution after a hypernode is exited. We defined a WF-CML hypergraph as follows:

$$\texttt{WF-CML\_HyperGraph} = \{\texttt{HyperNodes, HyperEdges, HyperEdgeAnn}\}$$

where:

$\texttt{HyperNodes} = \{\texttt{InitialNode}\} \cup \{\texttt{FinalNode}\} \cup \texttt{CS\_ProcNodes} \cup \texttt{WaitNodes}$,

`CS_ProcNodes` - a set of atomic communication process nodes.

`WaitNodes` - a set of wait nodes for handling synchronization for a join.

$\texttt{HyperEdges} \subseteq \texttt{HyperNodes} \times \texttt{HyperNodes}$

$\texttt{HyperEdgeAnn}: \texttt{HyperEdges} \rightarrow \texttt{EdgeAnn}$

`EdgeAnn` - events required to enable a transition.

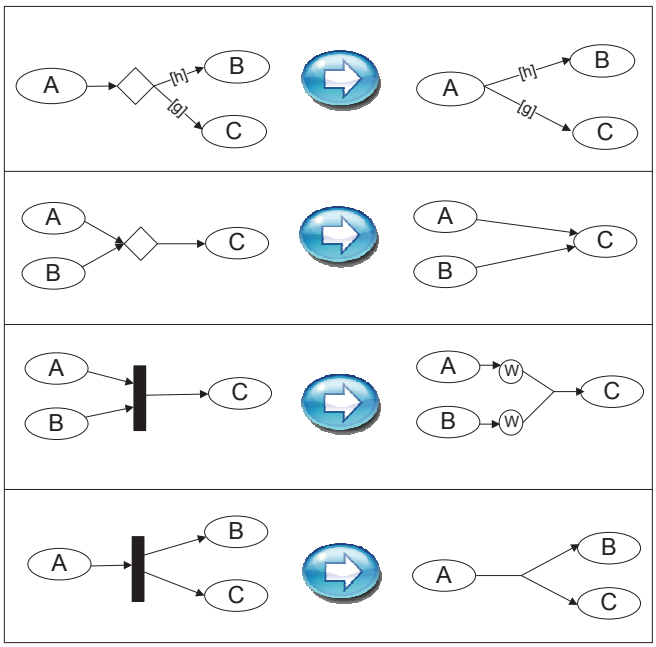

Figure 4.3: Elimination of Control Nodes in Generating Hypergraphs

Table 4.1: Mapping Control Nodes to Hyperedges

Control Nodes	*# of Hyperedge*	*# of Source nodes*	*# of target nodes*	*Has labels*
Decision	N	1	1	yes
Fork	1	1	N	no
Join	1	N (wait node)	1	no
Merge	N	1	1	no

Next, we define the detailed execution semantics of workflow models through the notion of WF-CML hypergraphs in terms of a two step process:

Mapping WF-CML process to WF-CML hypergraphs: The algorithm for converting WF-CML models into WF-CML hypergraphs is similar to the transformation of UML Activity Diagrams to Activity Hypergraphs designed by Eshuis et al. in [24]. The main steps are: (1) flattening the hierarchy by elimination of nested WF-CML models, (2) replace the join node with wait nodes, (3) combine edges of control nodes

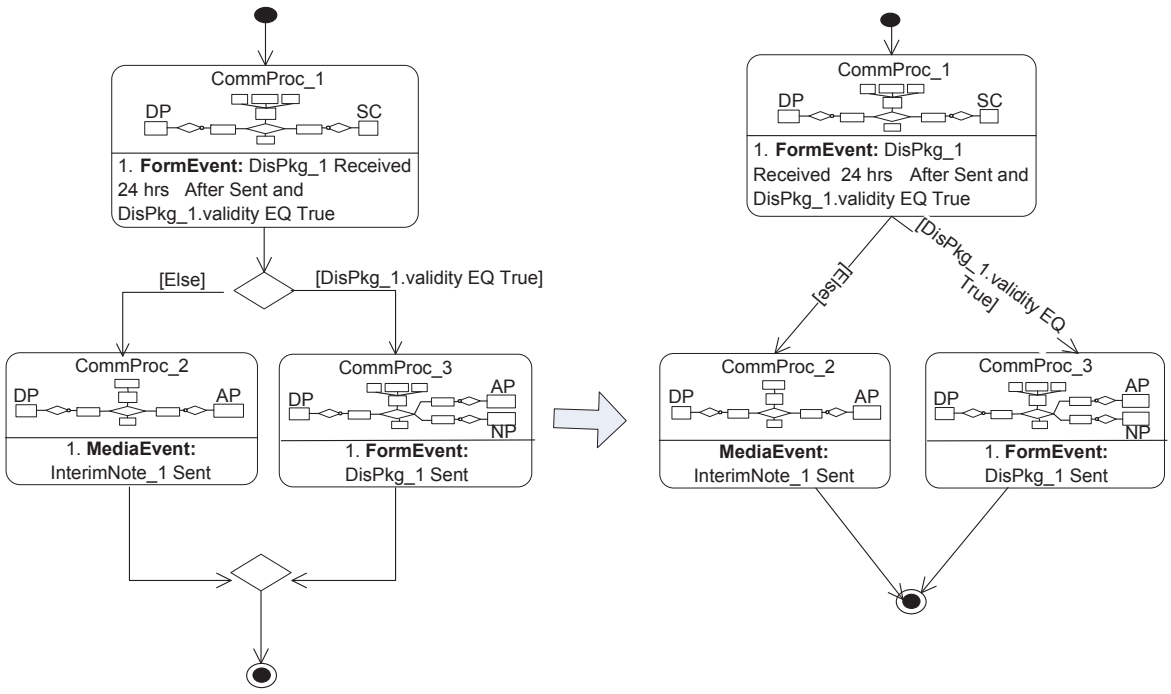

Figure 4.4: Converting WF-CML Model to WF-CML Hypergraph

to create hyperedges, including concatenation of edge annotations. The resulting activity hypergraph would essentially be a collection of communication activities, wait nodes, as well as initial and final nodes, connected by composite transitions, termed hypergraph. Note that each hyperedge is essentially a chain of transitions that are atomic: they are executed at the same time in the sense that it is not possible to execute the first part of the transition and then put it on hold. The mapping rules for converting control nodes is summerized in Table 4.1 In Figure 4.1(b), this mapping is done by the WF-CML Analyzer process.

To illustrate the mapping process, we show a simple example of how a WF-CML program is converted to a WF-CML hypergraph in Figure 4.4. Figure 4.4 shows now how a WF-CML model is mapped to its corresponding hypergraph.

Stepwise execution of Communication Process Nodes: After getting the WF-CML hypergraph, we would do a step-wise event-driven execution of the communication processes according to the prescribed order. We define the state space of a communication workflow system and use two algorithms to represent how the system behaves given certain inputs. Note that similar to other semantic specifications proposed for reactive systems, we also incorporate information about the environment in the definition of the system state space, including the input events, etc. In the next subsection, we formalize the semantics of WF-CML models using labeled transition systems, defined as extensions of the semantics for CML.

4.2 Towards a Formal Approach For Dynamic Synthesis

As previously mentioned, the semantic specifications of WF-CML are built on top of the semantic specifications for CML. In this subsection, we first formalize the semantics for CML models [77], and then extend it for WF-CML that provides the basis for dynamic synthesis of coordinated communication services.

4.2.1 Formal Semantics Specification for CML

We formalize the behavioral model of dynamically synthesizing CML models using a labeled transition system (LTS) defined in terms of states (or configurations) stored in the SE environment, and a transition relation. A LTS consists of a collection of states and a collection of state transitions between them. The transitions are specified by a labeled transition relation. This behavioral model is an extension and refinement of the work presented by Wang et al. [77]. The transition relation essentially specifies the rules for state changes based on the input CML model and the SE environment. It also defines the mapping from the input CML model to the output communication

control script. The transition relation (R_{Syn}) is defined as follows:

$$((CI_{in}, DI_{in}), Env_i) \Longrightarrow ((CI_{out}, DI_{out}), (Script_{Neg}, Script_{MT}), Env_{i+1}) \quad (4.1)$$

where:

- (CI_{in}, DI_{in}) - input control and data instance pair to be processed(see left side of Figure 4.1(b)).
- Env_i - current environment including the complete user's view of the communication schema instance currently being executed ($userSchema$) and a connection environment ($Conn\_Env_i$) for each of the connection processes currently being executed. The state of each connection, $Conn\_Env_i$, is defined as a two-tuple (Neg_i, MT_i), where:
  - Neg_i - current environment of a specific connection process with respect to (re)negotiation and it includes the control instance (CI_i) currently being negotiated or executed ($CI_i \in \{CI_{Neg}, CI_{exe}\}$), negotiation state ($Neg\_State$), number of received responses ($num\_responses$) and the presence of a negotiation token ($hasNegToken$). Neg_State is maintained in the `Negotiation` state machine discussed in Table A1.
  - MT_i - current environment of a specific connection process with respect to media transfer and it includes the currently executing DI (DI_{exe}), media transfer state (MT_State), and list of streams ($num\_streams$). MT_State is maintained in the `Media Transfer` state machine discussed in Table A2.
- (CI_{out}, DI_{out}) - output instance pair generated during the transition process. This pair contains the CI and DI that may be sent to the UCI representing the

currently executing connection process. The UCI updates the user's view of the communication using this instance pair.

- $(Script_{Neg}, Script_{MT})$ - control script pairs generated for the UCM to invoke the execution of the negotiation and media transfer macros. Note that these scripts may contain control instances and data instances for the remote participants in the communication. (see the rightmost column of Figure 4.1).

- Env_{i+1} - updated environment after the most recent transition. The structure is similar to Env_i stated above.

The behavior model associated with a connection is based on a sequence of instance pairs of the form (CI_i, DI_i), where $i = 0, 1 \cdots n$, CI is a control instance and DI a data instance. The initial instance pair (CI_0, DI_0) represents the initial state of the system with respect to some new connection to be established.

Then the incoming CI_1 is compared to CI_0 resulting in changes to the system state and the negotiation process initiated. The future behavior of the system is determined based on subsequent incoming instance pairs and the current state of the system.

The input instance pair (CI_{in}, DI_{in}) may contain the user's specifications for multiple connections. Each instance pair is broken down into a set of instances per connection and handled separately. For each connection, we break down the transition relation R_{Syn}, defined in equation (4.1), into a relation for handling control instances during negotiation, and one for handling data instances during media transfer. The following relations are defined per connection j:

$$(CI_{in}, Env_i.Conn_j.Neg_k) \Longrightarrow (CI_{out}, Script_{Neg}, Env_{i+1}.Conn_j.Neg_{k+1}) \quad (4.2)$$

$$(DI_{in}, Env_i.Conn_j.MT_k) \Longrightarrow (DI_{out}, Script_{MT}, Env_{i+1}.Conn_j.MT_{k+1}) \quad (4.3)$$

R_{Neg}, shown in equation (4.2), represents the relation transforming the control instance during negotiation, and R_{MT}, equation (4.3), the relation for transforming the data instance during media transfer. A key part in our semantic definition is the changes identified between two successive models, therefore we define the functions $analyze_{CS}$ and $analyze_{DS}$ that identify the changes in the control schema instances and data schema instances, respectively. These changes are used to trigger events in the labeled transition systems for negotiation and media transfer. To specify the transition relations, R_{Neg} and R_{MT} in more detail, we define the following functions :

$$analyze_{CI} : ControlSchema \times EnvNeg \rightarrow ControlChangeType \quad (4.4)$$

$$analyze_{DI} : DataSchema \times EnvMT \rightarrow DataChangeType \quad (4.5)$$

$$update_{Neg} : ControlSchema \times EnvNeg \times ControlChangeType \rightarrow ControlSchema \times EnvNeg \times NegScript \quad (4.6)$$

$$update_{MT} : DataSchema \times EnvMT \times DataChangeType \rightarrow DataSchema \times EnvMT \times MediaScript \quad (4.7)$$

where:

- *ControlSchema* is the type for control instances, CI (CI_{in} or CI_{out}), or the CI stored in the environment ($Neg_i.CI_{Neg}$)
- *DataSchema* is the type for data instances, DI (DI_{in} or DI_{out}) or DI stored in the environment($MT_i.DI_{exec}$)
- *ControlChangeType* and *DataChangeType* are the enumerated types for changes related to the input CI or DI, respectively
- *EnvNeg* and *EnvMT* are the types for the negotiation environment (Neg_i) and media transfer state (MT_i), respectively in $Conn\_Env_j$
- *NegScript* and *MediaScript* are the types for the control scripts generated for negotiation ($Script_{Neg}$) and media transfer ($Script_{MT}$), respectively

The transition relation R_{Neg} can be defined by $analyze_{CI}$ in equation (4.4) and $update_{Neg}$ in equation (4.6) as follows:

$$R_{Neg} = update_{Neg}(CI_{in}, Env_i.Conn_j.Neg_k, analyze_{CI}(CI_{in}, Env_i.Conn_j.Neg_k)) \tag{4.8}$$

Similarly, we specify the translation relation R_{MT} as follows, using $analyze_{DI}$ in equation (4.5) and $update_{MT}$ in equation (4.7) :

$$R_{MT} = update_{MT}(DI_{in}, Env_i.Conn_j.MT_k, analyze_{DI}(DI_{in}, Env_i.Conn_j.MT_k)) \tag{4.9}$$

Relating the above definition with the execution flow shown in Figure 4.1(b), the Env_i is maintained in the `Comm.Schema Controller`, which also realizes synthesis

algorithms including R_{Syn}, R_{Neg} and R_{MT}. The negotiation and media transfer states are maintained by the `negotiation` and `media transfer` processes, respectively. The specifics of these functions are explained further in terms of synthesis algorithms and labeled transition systems in [82]. Although it is not the major contribution of this dissertation, for complete reasons, we show synthesis algorithms for synthesizing CML models in Appendix 6.1, analyzing CML control and data schema instances in Appendix 6.2, 6.3, and the details of the negotiation and media transfer state machines in Table A1,A2 in the Appendix.

4.2.2 Formal Semantics Specification for WF-CML

We formalize the behavioral model of dynamically synthesizing WF-CML models by extending the labeled transition system (LTS) defined previously for CML. This is done through expanding the state space (all possible configurations) stored in the SE environment, and designing new transition relations based on existing transition rules. More specifically, the following steps are needed:

1. Formalize the state space of collaborative communication systems at runtime
2. Define auxiliary functions to query/modify the system runtime state defined in step 1
3. Specify transition relations to realize the execution of the communication-specific workflow models, through the use of auxiliary functions defined in step 2

Next we present our approach to formalize the semantics of WF-CML models based on the semantics of CML models:

1. Modeling the environment of the WF-CML runtime using a 4-tuple:

$\text{WF\_Env}_i \Longrightarrow (\text{WF}_{exec}, \text{Comm\_Env}, \text{CurNodes}, \text{WF}_{input})$, in which:

- WF_{exec} - the currently executing WF-CML hypergraph in this WF_Proc process.
- Comm_Env - a list of executing CS processes in the executing WF_Proc process.
- Cur_Nodes - currently active CS process nodes or wait nodes with respect to the WF_Proc process.
- WF_{input} - inputs that trigger the progress of collaborative systems:

In the above definition, we have covered the syntactical description of the WF_{exec} and the definition for Comm_Env in previous sections. Here, we focus on the definition for Cur_Nodes and WF_{in} next.

$\text{Cur\_Nodes} \subset \text{Set}\langle\ \text{CS\_ProcNodes}\ \rangle \cup \text{WaitNodes}$, in which,

- CS_ProcNodes represent a set of atomic communication process nodes
- WaitNodes and a set of wait nodes for handling synchronization for a join in a WF-CML hypergraph separately, as mentioned previously

$\text{WF}_{input} \Longrightarrow \text{WF\_Event}_{in} \cup \text{WF\_Update}_{in}$, in which:

- WF_Event_{in} - an input event that may trigger the execution of the next node in the WF-CML model. These events include negotiation events, data transfer events and exception events.
- WF_Update_{in} - an updated workflow model passed down from the user interface, a trivial case might be a new communication schema wrapped up in a trivial workflow model

2. Defining auxiliary functions for inspecting and modifying the system state. We identified the following auxiliary functions and grouped them into *mutator* functions (functions that modify the system state) and *accessor* functions (functions that only inspect or query the system state). These functions facilitate the specification of transition relations in the next step.

Mutator functions:

- addComm → create a new communication process
- updateComm → update an existing communication process
- next → return the next node of an existing node

Accessor functions:

- isTrigger → check if an input event is a trigger event
- checkEquivProc → check if the current configuration of a process is semantically equivalent with its initial configuration
- eval → evaluate the truth value of a boolean expression
- next → return the next node of a given node
- enabled → return the set of enabled edges given the current environment
- isWaitNode → check if a given node is a wait node or not
- num → return the number of elements in a given set

Formally, these auxiliary functions are defined as mappings from the domain to its image as follows:

$$addComm : ControlSchema \times EnvNeg \rightarrow EnvNeg \tag{4.10}$$

$$updateComm : ControlSchema \times EnvNeg \rightarrow EnvNeg \tag{4.11}$$

$$isTrigger : WF\_Event \rightarrow \{True, False\} \tag{4.12}$$

$$checkEquivProc : CommProcNode \times EnvNeg \rightarrow \{True, False\} \tag{4.13}$$

$$eval : EnvNeg \rightarrow \{True, False\} \tag{4.14}$$

$$next : EnvNeg \rightarrow CommProcNode \tag{4.15}$$

$$enabled : WF\_Event \times EnvNeg \rightarrow HEdge \tag{4.16}$$

$$isWaitNode : CS\_ProcNodes \cup WaitNodes \rightarrow \{True, False\} \tag{4.17}$$

$$num : Set \rightarrow Integer \tag{4.18}$$

3. Specify the transition relation for executing communication-specific workflow models, including advancing the process model as well as updating communication service configurations. We first give a brief explanation of the identified six transition relations, and then formalize each relation separately.

- $R_{addComm}$ - represents the addition of a new communication
- $R_{updateComm}$ - represents the update of an existing communication
- $R_{triggerComm}$ - represents triggering the communication process to the next node
- $R_{choiceComm}$ - represents the choice of different paths in a communication process
- $R_{concurComm}$ - presents parallel paths in a communication process
- $R_{syncComm}$ - presents the synchronization of parallel paths in a process

We then formalize these transition relations using set theoretic notations to represent semantic concept [52]. Note that we use s and s′ to represent the states before and after the transition, respectively. Two states s, s′ are related by the transition if and only if the system can change state from s to s′. We use the notation *s.** to access specific components of the state definition, and use the form s[x *mapsto* a_0] to represent a new state in which each occurrence of a variable x is replaced with another expression a_0. This is called substitution in [52] to represent the semantics

of expressions. Also, we use OCL collection operations such as select to iterate and filter through set elements.

$$R_{addComm} \overset{df}{\Leftrightarrow} \neg\ \exists\ \text{n} \in \text{s.comm\_env} \bullet \text{ci}_{in}.\text{cid} = \text{n.cid} \\ \wedge\ \text{s}\prime = \text{s}[\text{comm\_env} \mapsto \text{addComm}(\text{ci}_{in}, \text{s})] \tag{4.19}$$

$$R_{updateComm} \overset{df}{\Leftrightarrow} \exists\ \text{n} \in \text{s.comm\_env} \bullet \text{ci}_{in}.\text{cid} = \text{n.cid} \\ \wedge\ \text{s}\prime = \text{s}[\text{comm\_env} \mapsto \text{updateComm}(\text{ci}_{in}, \text{s})] \tag{4.20}$$

$$R_{triggerComm} \overset{df}{\Leftrightarrow} \exists\ \text{e} \in \text{s.wf\_event}\ \exists\ \text{n} \in \text{s.cur\_nodes} \bullet \text{isTrigger(e, n)} \\ \wedge\ \text{num(n.edges)} = 1 \wedge \text{num(enabled(e, s))} = 1 \\ \wedge\ \text{s}\prime = \text{s}[\text{cur\_nodes -}\{\text{n}\} + \text{next(s, n)}] \tag{4.21}$$

$$R_{choiceComm} \overset{df}{\Leftrightarrow} \exists\ \text{e} \in \text{s.wf\_event}\ \exists\ \text{n} \in \text{s.cur\_nodes} \bullet \text{isTrigger(e, n)} \\ \wedge\ \text{num(n.edges)} > 1 \wedge \text{num(enabled(e, s))} = 1 \\ \wedge\ \text{s}\prime = \text{s}[\text{cur\_nodes - }\{\text{n}\} + (\text{n.edges.select}\{\text{e}|\ \text{eval(e,s)}\ \})] \tag{4.22}$$

$$R_{concurComm} \overset{df}{\Leftrightarrow} \exists\ \text{e} \in \text{s.wf\_event}\ \exists\ \text{n} \in \text{s.cur\_nodes} \bullet \text{isTrigger(e,n)} \\ \wedge\ \text{num(n.edges)} > 1 \wedge \text{num(enabled(e, s))} > 1 \\ \wedge\ \text{s}\prime = \text{s}[\text{cur\_nodes - }\{\text{n}\} + \text{next(n,s)}] \tag{4.23}$$

$$R_{syncComm} \overset{df}{\Leftrightarrow} \exists\ \text{n} \in \text{s.cur\_nodes} \bullet \text{isWaitNode(n)} \wedge \forall\ \text{n}\prime \in \text{n.edges.srcs}$$
$$\bullet\ \text{isWaitNode(n}\prime) \wedge \text{n}\prime \in \text{s.cur\_nodes} \tag{4.24}$$
$$\wedge\ \text{s}\prime\text{=s[cur\_nodes - n.edges.srcs + next(n,s)]}$$

As mentioned earlier, the transitions relations make use of the auxiliary functions defined previously to query about the system state as well as to affect changes in the system states as a result of the transition. Next we present synthesis algorithms that realize this semantics formalism by taking an instance of a WF-CML model, acting upon it based on the runtime state of the system and affecting the system state at the same time.

4.2.3 Synthesis Algorithms For Realizing WF-CML

To realize the semantics specifications described in the previous section, we now describe synthesis algorithms that take a WF-CML model instance, analyze it and execute the communication services and control behavior specified in the instance. We present a top level realization algorithm, a WF-CML analysis algorithm, and a detailed execution algorithm that traversals the WF-CML hypergraph to realize the required service coordinations.

Realization: The top level algorithm realize_WF, shown in Algorithm 4.1, is the entry point for any input WF-CML model. It invokes the algorithm analyze_WF, line 2, to dynamically analyzing WF-CML models. The analyze_WF is shown in Algorithm 4.2. Based on the result of the analysis, realize_WF either instantiates a new process, *wf_proc*, to handle the realization of a new WF-CML model, lines 3-6; delegates the realization of a new or existing CS process to the controller for the communication

Algorithm 4.1 Algorithm to synthesize WF-CML.

```
1: realize_WF (ref WF_in)
   /*Input: WF_in - WF-CML model */
2: wfc ← analyze_WF(WF_in, WFProcs)
3: if wfc.diff == "Initial" then
4:     wf_proc ← new WFProc(wfc.WF_HPG)
5:     WFProcs.add(wf_proc)
6:     wf_proc.Exec_HPG(wf_hpg, WF_Env_i)
7: else if wfc.diff == "CS_ProcUpdate" then
8:     CS_Controller.execute(wfc.cmlSchema)
       /*Extract the new (CI, DI) and updates the executing comm. instances. Communication
       control scriptsare generated during updates to the comm. instances */
9: else if wfc.diff == "WFUpdate" then
10:    wf_proc ← WFProcs.find(wfc)
11:    wf_proc.update(wfc, WF_Env_i)
       /* Update the executing WF-CML model using wfc */
12: else if wfc.diff == "Terminate" then
13:    wf_proc ← WFProcs.find(wfc)
14:    wf_proc.terminate(WF_Env_i)
15: end if
```

schema [77], lines 7-8; handles dynamic updates to the WF-CML model, lines 9-11; or terminates the currently executing WF-CML model, lines 13-16. Instantiating a new WF-CML model requires the execution of the WF-CML hypergraph, line 6, returned from analyze_WF.

Algorithm analyze_WF performs a runtime analysis of the incoming WF-CML model and returns a workflow change object (*wfc*). The *wfc* object contains three fields: *diff* containing the change type, *WF-HPG* the WF-CML hypergraph, and a CML schema pair (CI, DI). If the input WF_{in} contains a new WF-CML model, lines 2-4, then a WF-CML hypergraph is built and returned as a part of *wfc*. If WF_{in} contains a trivial WF-CML model (a model with one CS process) then either it signals the termination of the workflow, lines 6-7, or an update to a CS process, lines 8-10. The update to the CS process results in the CML model being extracted from WF_{in} and returned as part of *wfc*, line 10. If WF_{in} is an update to an existing WF-CML model, lines 12-15, a new hypergraph is created a returned in *wfc*.

Algorithm 4.2 Algorithm to analyze WF-CML.

```
1: analyze_WF (ref WF_in, ref WFProcs)
   /*Input: WF_in - WF-CML model
    Output: wfc - WF-CML change object */
2: if !WFProcs.contains(WF_in) then
3:    wfc.diff ← "Initial"
4:    wfc.WF_HPG ← Map2HPG(WF_in)
      /* An initial WF-CML model */
5: else if WF_in.isTrivial() then
6:    if WF_in.csProcNode.isEmpty() then
7:       wfc.diff ← "Terminate"
         /*Model has no workflow related nodes */
8:    else
9:       wfc.diff ← "CS_ProcUpdate"
         /* Update to a CS process */
10:      wfc.cmlSchema ← WF_in.csProcNode.commSchema
11:   end if
12: else
13:   wfc.diff ← "WFUpdate"
14:   wfc.WF_HPG ← Map2HPG(WF_in)
      /* Update to the flow in WF-CML model */
15: end if
16: return wfc
```

Analysis of Algorithm: The running time of `realize_WF` shown in Algorithm 4.1 is dependent on the running time of `analyze_WF`. Although there are different algorithms for converting WF-CML models to WF-CML hypergraphs, here we focus on analyzing the worse case running time. Using a naive implementation, the running time for converting WF-CML models to hypergraphs depends on the number of nodes in the model, as well as the maximum degree (in-degree or out-degree) of each node. We deduce that the worst case running time for the algorithm `synthesisCML` is $O(\|N\| * D))$ where N is the number of nodes in the WF-CML model, and D is the maximum degree of all nodes in the model. Since in most cases, the maximum degree of the nodes will be a relatively small number, the worst case running time could be approximated to $O(\|N\|)$.

Execution of WF-CML Hypergraphs: The execution of a WF-CML hypergraph involves traversal of the hypergraph supported by the underlying event mechanism. The approach we use is similar to that described by Eshuis et al. [24] except for the

following differences: (1) the restrictions we place on the trigger events in the CS processes and the guards annotating the hyperedges, and (2) the ability for a CS processes to continue execution after a hypernode is exited. We present an explanation of the execution algorithms for the hypergraph traversal and focus on the differences previously stated.

Execution algorithm: Inspired by Eshuis et al. in [24], we designed the execution algorithm for the traversal of the WF-CML hypergraph. Besides initialization, the algorithm mainly involves a loop that repeatedly execute the following steps:

1. Pick an event from the event queue in the system runtime
2. Compute a step based on the event and the system runtime
3. Execute the step while updating the event queue and current system state

Details of this algorithm is shown in Algorithm 4.3. As we see, line 2-5 initializes the environment of this WF-CML process, and starts the execution of the first communication service specified in the model. Then, line 6-16 details the loop for the repeatedly traversal of the WF-CML hypergraph based on events received and the current state of the WF-CML process. Note that the calculation of enabled edges in line 6 is detailed out in the Eval_WF_Step algorithm. For each of the enabled edges that is returned from the algorithm, we replace the source nodes of the edge with target nodes (note that both source and target nodes could be multiple), as shown in line 10-11. Then we execute the services in each of the target nodes by passing it to the communication schema controller.

The Eval_WF_Step algorithm works in a straightforward way: it first calculates the nodes that are triggered to be exited. And for each of these nodes, it basically considers three cases: (1) one outgoing edge with one source (sequence or merge),

line 25-26; (2) one outgoing edge with multiple source (fork), line 27-28; (3) multiple outgoing edges (join), line 31-35. The set of enabled edges are then returned to the Exec_WF_HPG for further execution.

Next we introduce how events are being generated and confused in the synthesis process. Also how advanced control concepts like looping are currently being handled and will be handled in the future. Event mechanism: In Section 3.3, we classify atomic events into data transfer events, negotiation events, exception events, and so on. These events could either be generated externally, or internally. Given that the current design of WF-CML allows for the concurrent execution of multiple WF-CML processes, a central event manager is needed to receive events, queue them and dispatch them to corresponding WF-CML processes. The event manager is also responsible for generating timeout events. Here, we assume the events that a CS process is waiting for will eventually arrive. In addition, since the trigger event for a CS process is based on events generated by the executing CS processes in the active hypergraph node, events that belong to a future CS process is ignored. Our design of WF-CML ensures that if the trigger event of a CS process fires then there is at least one edge that can be taken out of the node. This is more restrictive than the semantics defined by Eshuis et al. [24] for UML activity diagrams.

Handling Loops: When a WF-CML model contains a loop that connects a CS process node to a previous CS process that is still executing, there is an option of restarting a new CS process, or reusing the currently executing CS process. To make a decision, we first need to define the notion of equivalence of CS processes. Recall that a CS process consists of an executing communication instance pair (CI, DI) and a trigger event. At this stage in the development of WF-CML we define equivalence only on the CI. We currently define "total equivalence" of CI based on the attributes of, and number of, the Connections, Persons, MediumTypes and FormTypes. For example,

Algorithm 4.3 Algorithm to Execute WF-CML Hypergraph.

```
1: Exec_WF_HPG (ref wf_hpg, ref WF_Env_i)
   /* Method to execute the WF hypergraph */
   /* Input: wf_hpg - WF hypergraph to be executed
           WF_Env_i - current workflow environment */
2: enabled ← wf_hpg.initialNode.outEdge
3: WF_Env_i.CurNodes.add(enabled.targetNode)
4: schema ← enabled.targetNode.schema
5: CommSchemaController.execute(schema, WF_Env_i.Comm_Env)
6: while true do
7:    wfEvent ← queue.nextEvent()
      /* wait for input events from environment */
8:    enabled ← Eval_WF_Step(WF_Env_i, wfEvent) /* compute a super step */
9:    for each edge ∈ enabled do
10:      WF_Env_i.CurNodes.add(edge.targets)
11:      WF_Env_i.CurNodes.remove(edge.srcs)
12:      for each node ∈ edge.targets do
13:         CommSchemaController.execute(node.schema, WF_Env_i.Comm_Env)
14:      end for
15:   end for
16: end while

17: Eval_WF_Step (ref WF_Env_i, wfEvent)
   /* method to evaluate and return a set of enabled edges */
   /* Input: WF_Env_i - current state of SE environment, wfEvent - external event */
   /* Output: edges - set of enabled edges */
18: for each n ∈ WF_Env_i.CurNodes do
19:   if wfEvent ∈ n.triggerEvents then
20:      nodeToLeave ← n
21:   end if
22: end for
23: if nodeToLeave.outEdges.size() == 1 then
24:   oe = nodeToLeave.outEdges.any
      /* when the hyperedge has one source node, it is directly enabled */
25:   if oe.sources.size()== 1 then
26:      return oe
27:   else if oe.sources.size()> 1 && oe.sources.forAll{s| s ∈ WF_Env_i.CurNodes} then
28:      return oe
29:   end if
30: else
31:   for all oe ∈ nodeToLeave.outEdges do
32:      if oe.eval() == true then
33:         return oe
34:      end if
35:   end for
36: end if
```

starting with Connections: the number of connections must be equal, the number of attached devices, medium types and form types for each connection must also be equal. Similarly we check equality for Persons, MediumTypes and FormTypes. We expect to relax this strict definition of equivalence of CIs in the future.

Analysis of Algorithm: The algorithm for executing WF-CML hypergraphs is an interactive process which depends on real-time information such as the arrival rate of the events. It also depends on runtime information such as the number of current nodes ($\|N\|$), the number of enabled edges($\|E\|$), and the number of target nodes for each enabled edge ($\|D\|$) at any point in time. Since in most cases, the maximum degree of the nodes will be a relatively small number (constant), the worst case running time could be approximated to $O(\|N\| + \|E\|)$.

4.3 Validation of Dynamic Synthesis

In this section, we validate the behavioral semantics of WF-CML models, described in Section 4.2, using Kermeta [69] as our simulation framework. We simulated several scenarios from a cross-section of use cases using our implementation, however we only present the result of a scenario for the healthcare use case described in Section 3.2.1.

4.3.1 Model Simulation Using Kermeta

Choice of Simulation Tool: The goal of our simulation is to validate the correctness of the operational semantics of WF-CML models for dynamic synthesis. Therefore, a meta-modeling tooling facility is needed that could help us define WF-CML both syntactically and semantically. In addition, a meta-modeling facility that allows us to define executable models would be very helpful. As a result of a comparison of several

meta-modeling frameworks, we chose Kermeta [69], an executable meta-modeling language with a full-fledged workbench environment, as our simulation framework. Kermeta [69] allows description of metamodels whose models are executable. The characteristics of Kermeta are as follows:

- It is a powerful meta-programming environment for engineering domain specific languages like WF-CML, with strong tooling support
- It allows the description of meta-models whose models are directly executable
- It is easy to use due to the extensive documentation available online

Simulation Approach: Kermeta program specifies the meta-model of domain specific languages conforming to the EMOF standard. Moreover, actions of the language constructs are specified as class operations in the metamodel. Kermeta is integrated as a plug-in to the Eclipse IDE, and it provides a generation tool Ecore2Kermeta which has allowed us to translate our WF-CML metamodel (Figure 3.2(a)) to a Kermeta compatible version. An example of a ECORE file before converting to a Keremta version is shown in Figure 4.5

In our simulation, we first specified the WF-CML metamodel using ECORE and used the generation tool Ecore2Kermeta in the workbench to translate the meta-model (Figure 3.2) to a Kermeta compatible version. Then we coded the semantics specification defined in Section 4.2 in the Kermeta language. As an example of how Kermeta is used to specify executable metamodels, a piece of Kermeta code is shown in Figure 4.6 and in Figure 4.7. We see that operations like execute provides the execution logic for the CommProcessNode.

In order to simulate the dynamic synthesis of WF-CML models we created a harness which included stubs and drivers. The stubs included a pretty printer that

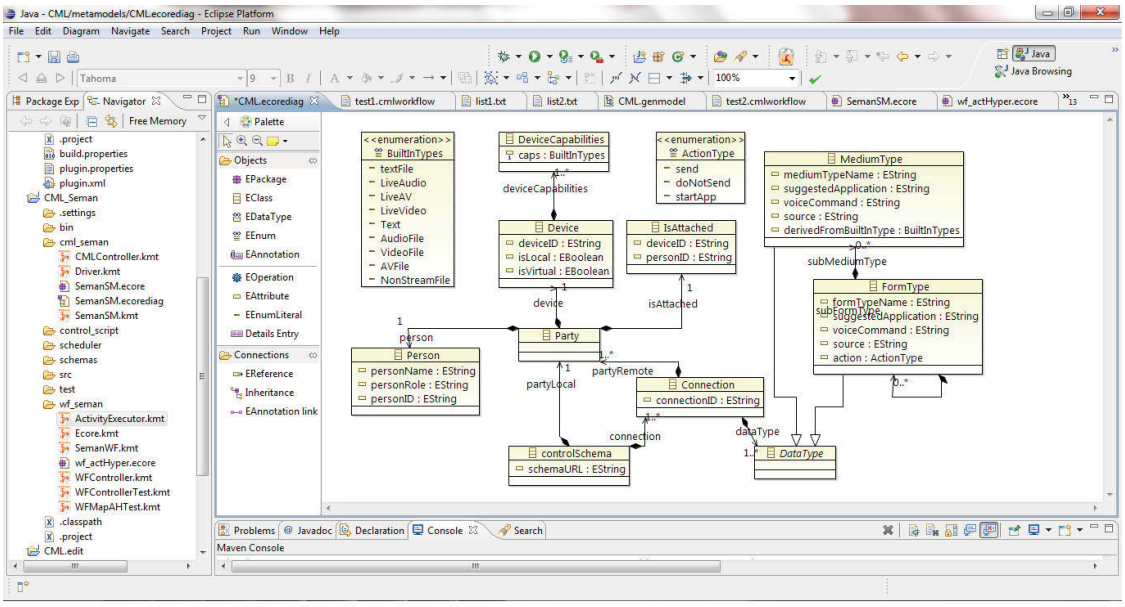

Figure 4.5: A Metamodel Ecore File Before Conversion to Kermeta File.

generated control scripts, as well as updated models for the user. A test driver was also created, which basically initiated a workflow controller object (WF-CML controller), loaded a WF-CML model instance and passed it to the workflow controller. In terms of events that trigger the workflow execution, two ways of feeding events into the workflow controller were designed: (1) A set of predefined events that triggered the workflow progression were provided to the workflow controller all at once before execution. (2) Having the user input events on-the-fly at runtime. Note that workflow events generated at runtime during execution could also be fed back into the workflow controller for further execution. At runtime, when a WF-CML model instance is loaded into the environment, Kermeta will execute the model (an instance of the WF-CML metamodel) based on the specified behavioral semantics in the metamodel. As a result of the WF-CML model execution, target model instances (communication control scripts) are generated, as well as feedbacks to the user.

Limitations of the Simulation: One limitation is the current event mechanism due to Kermeta's lack of support for generating temporal events, timeout events like

```
1  aspect class CommProcessNode  inherits FlowNode {
2     attribute cmlSchema : CommSchema[1..1];
3     attribute owner : ecore::EString;
4     attribute triggerEvent : Event[1..*];
5     attribute processName : ecore::EString;
6     attribute status : procStatus;

7     operation execute(procExcutor:CommProcExecutor,
8          queue:BlockingQueue<Event>): Boolean  is do {
9       status := procStatus.active;
10      stdio.writeln("Process"+processName+"executing");
11      result:= procExcutor.executeCommProc
12                    (cmlSchema.controlSchema, queue);
13    end}
14    operation re_execute(newSchema:Object,procExcutor:
15        CommProcExecutor,queue: BlockingQueue<Event>):
16      Boolean is do {
17      stdio.writeln("Still in Process "+ processName);
18      result:= procExcutor.executeCommProc
19                                  (newSchema,queue);
20    end}
21    operation deactivate(): Void is do {
22      status := procStatus.inactive;
23      stdio.writeln("Process"+processName+"inactive");
24    end}
```

Figure 4.6: Kermeta Code For CommProcessNode class

"After 24 hours" would have to be manually entered into the controller, instead of being automatically generated through a system clock. Another limitation is the concurrency mechanism in Kermeta. Due to Kermeta's indirect support for concurrent operations, we had to use an interleaving semantics for modeling concurrency between the Negotiation and Media Transfer during each communication process as well as inter-process concurrency. Interleaving semantics proved enough for the purpose of simulating WF-CML models.

4.3.2 Case Study

As part of the case study, we use the follow scenario that is modeled and simulated in the Kermeta framework. We show the WF-CML and event inputs that is fed as into the simulation tool that we built, and analyzed the execution trace of the simulator.

Scenario: On the day of discharge Dr. Burke (DP) creates a discharge package consisting of a text file (`RecSum-Jane.txt`), summary of patient's condition; and a

```
1  aspect class WFController {
2     attribute block_queue: BlockingQueue<Event>;
3     attribute fileio: FileIO;
4     reference csManager: CMLScheduler;
5     operation execute(ah: CommWorkFlowAH): Boolean  is do {
6        block_queue := BlockingQueue<Event>.new
7        stdio.writeln("WF Controller started");
8        var isEventsSeperate:  String init stdio.read
9        ("Will subsequent events be input seperately or all in one file?");
10       if(isEventsSeperate == "no") then
11         varinputs: Collection<String> init makeCollection(readFromStdIO())
12         inputs.each i| ah.queueEventsByID(block_queue, i)
13       end
14       ah.start(block_queue)
15     end}

16     operation getInstance(): WFController is do {
17       if (self == void)  or (csManager == void)  then
18         csManager := CMLScheduler.new
19         csManager.initialize()
20         csManager.local_user :=‘‘yali1028’’
21         fileio:= FileIO.new
22       end
23      result := self
24     end}

25     operation readFromStdIO():kermeta::standard::String is do {
26        var s : kermeta::standard::String
27        from var found: kermeta::standard::Boolean  init false ;
28        until found
29        /* code to  getting input file that has a list of input models */
30      end}
```

Figure 4.7: Kermeta Code For WFController class

video clip (`HeartEcho-Jane.mpg`), an echocardiogram (echo) of the patient's heart. Dr. Burke then establishes an audio video connection with Dr. Monteiro (SC) to discuss the patient's condition and shares the discharge package with him. During the conversation with Dr. Monteiro, Dr. Burke decides to contact Dr. Sanchez (PCP) (via an audio video connection) to join the conversation to discuss specific treatments. After the conference call is terminated Dr. Monteiro validates the discharge package and returns it to Dr. Burke who then sends it to Dr. Sanchez and the patient.

We used an incremental approach to simulate how the above scenario is realized in the CVM. First, we simulated the execution of basic communication processes such as the establishment of audio video connections, adding participants to join conversations as well as data transfer between participants in the same connection. The execution of each communication process is realized by the generated communication

Table 4.2: Applying the execution semantics to the healthcare scenario.

Row #	*Active Node*	*Received wfEvent*	*Is Trigger Event*	*Guard*	Target
1	Initial	-	-	None	Comm. Process 1: Exec(X-CML_1)
2	Comm. Process 1: Exec(X-CML_1)	NegComplete	N	-	-
3	Comm. Process 1: Exec(X-CML_Data1)	-	-	-	-
4	Comm. Process 1: Exec(X-CML_1)	MediaSent: Form_D1	N	-	-
5	Comm. Process 1: Exec(X-CML_1)	MediaReceived: Form_D1	Y	Form_D1 validated	Comm. Process 2: Exec(X-CML_2)
6	Comm. Process 2: Exec(X-CML_2)	NegComplete	N	-	-
7	Comm. Process 2: Exec(X-CML_2)	MediaSent : Form_D1	Y	None	WorkflowFinal
8	WorkflowFinal	-	-	-	-

Table 4.3: Applying the execution semantics to the healthcare scenario - cont.

Row #	*Active Node*	*Received wfEvent*	Generated Script
1	Initial	-	-
2	Comm. Process 1:	NegComplete	createConnections(”connection1”) sendSchema (”connection1” , ”burke” , ”monteiro”,”test1.cmlcontrolsyntax” , ””) addParticipants(”connection1”,”monteiro”)
3	Comm. Process 1: Exec(X-CML_Data1)	-	enableInitiatorMedia(”connection1”,”LiveAudio”)
4	Comm. Process 1: Exec(X-CML_1)	MediaSent: Form_D1	sendForm(”connection1”,”D1”,”RecSum-Jane.txt, HeartEcho-Jane.mpg”,”send”)
5	Comm. Process 1: Exec(X-CML_1)	MediaReceived: Form_D1	-
6	Comm. Process 2:	NegComplete	-
7	Comm. Process 2: Exec(X-CML_2)	MediaSent : Form_D1	sendForm(”connection1”,”D1”,”RecSum-Jane.txt, HeartEcho-Jane.mpg”, ”send”
8	WorkflowFinal	-	-

control script, which is then deployed on the UCM, a lower layer in the Communication Virtual Machine(CVM). Simulation results show that the correct sequence of control scripts are generated when the communication process executes the communication schemas.

To visualize the process of dynamically synthesizing CML models, we show a series of CML instances and the control scripts generated as a result of dynamic synthesis in Figure 4.8. The table has three columns: the source of input, G-CML representations of the X-CML instances, and the generated control scripts. The synthesis of these CML instances at runtime realized the scenario described in Section 3.2.1. For easy

readability, we only show a skeleton graphical version of the CML instance. We also break down the realization of communication services into three phases: initial negotiation of a two-way communication, enabling live audio/video, renegotiation into a three-way communication, and the media transfer of the patient discharge form. Note that SE also utilizes its environment during this process updating it accordingly, not shown in Figure 4.8. Then, we simulated the execution of WF-CML models that handle the coordination of basic communication processes. A WF-CML model instance is loaded by Dr. Burke and realized by the CVM. Table 4.2 shows the execution trace of the scenario generated by applying the algorithms in Algorithms 4.1 and 4.2. Columns 2 through 7 represent the active node, received wfEvent - event received by the workflow controller, a check for the trigger event, the guard, the target node and generated control script during the execution of the communication process. Each communication process will not move forward until the incoming event matches the trigger event condition. Incoming events are either an external event, like a MediaReceived event, or a system generated event, like "After 24 hours". Note that the communication schema contained within a process node could be updated, as show in Row 3 in Table 4.2. We do not show all possible events generated by the SE dispatcher due to space limitations.

To demonstrate the potentially different paths of execution for the scenario that we used, we simulated two different cases (1) the patient discharge package is received within 24 hours and is validated; (2) a time-out event occurs. The execution trace of the simulation shows that different paths are taken depending on the specific type of events received during the execution of the scenario. We show the output of the simulation trace in Appendix A13-A15 and Appendix A16-A18 . Note that Appendix A13-A15 show the execution trace when for the first case, while Appendix A16-A18 illustrate the other case.

4.4 Chapter Summary

In this chapter, we presented the dynamic semantics of WF-CML that is essential for the rapid realization of coordinated communication services. We started the semantic specifications with the introduction of the overall synthesis process, detailed labeled transitions systems, and synthesis algorithms for realizing the transition systems. The result of this semantics formalism is then simulated using the Kermeta metamodeling framework [69] for validation and verification of correctness. The simulation also serves the purpose of rapid language prototyping of WF-CML. In the next chapter, we introduce the *Communication Virtual Machine* prototype and related experimental studies of the usage of WF-CML using the prototype.

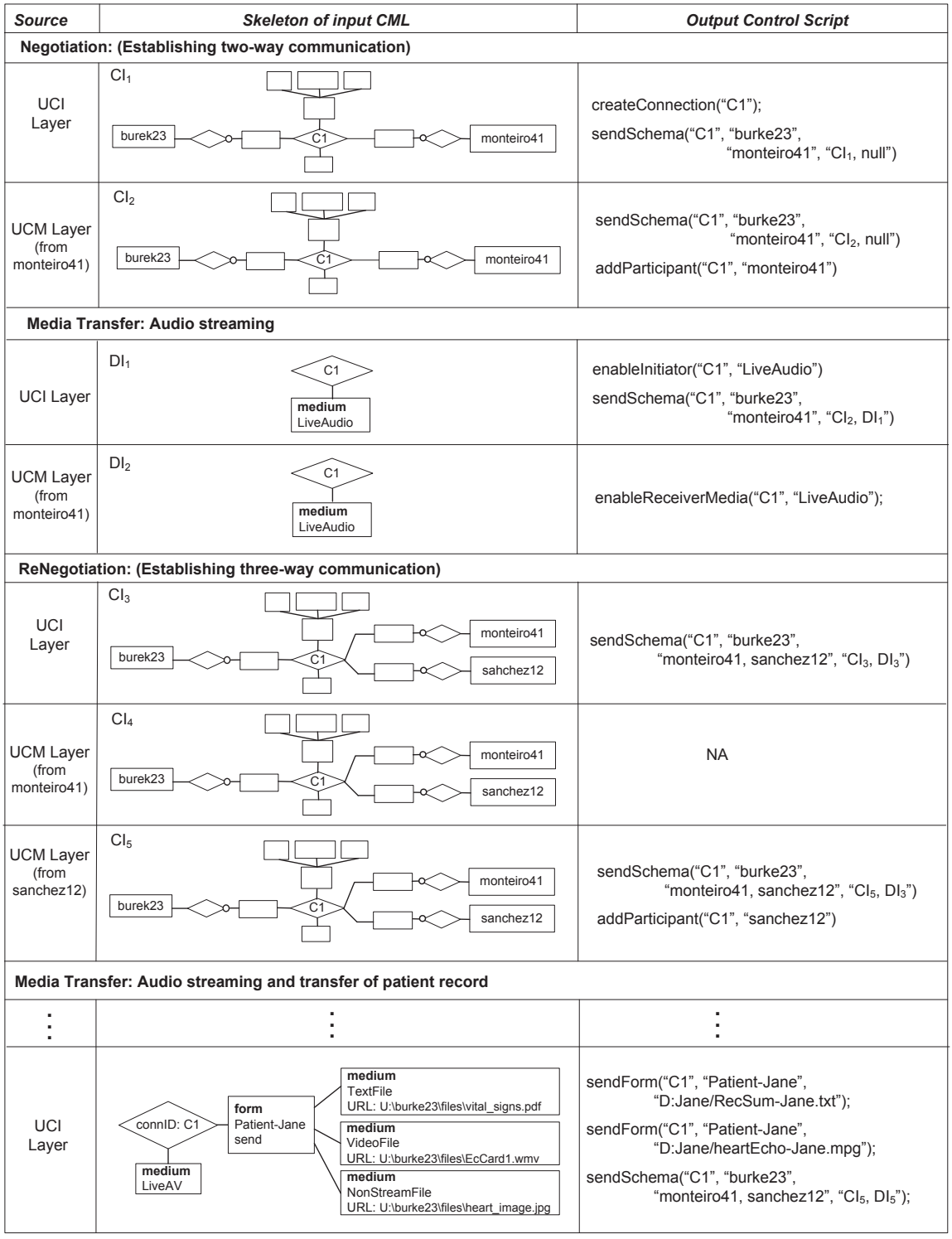

Figure 4.8: CML models and Output Control Scripts for Healthcare Scenario

CHAPTER 5

PROTOTYPE AND EVALUATION

In this chapter, we present details of the Communication Virtual Machine prototype, and measured static and dynamic metrics of the prototype. To show the advantages of using a DSML versus a general purpose workflow language, we also perform a comparative study between WF-CML and other workflow languages and measure the associated effort for the development process. The result of the comparative studies provides evidence on potential productivity gains of using WF-CML.

5.1 CVM Prototype

In this section, we explain the design and implementation of the Communication Virtual Machine (CVM) prototype that could dynamically synthesize, and execute coordinated UCCSs. First, the original implementation of the CVM prototype is introduced, then we introduce the extensions required to realize the coordination of communication models.

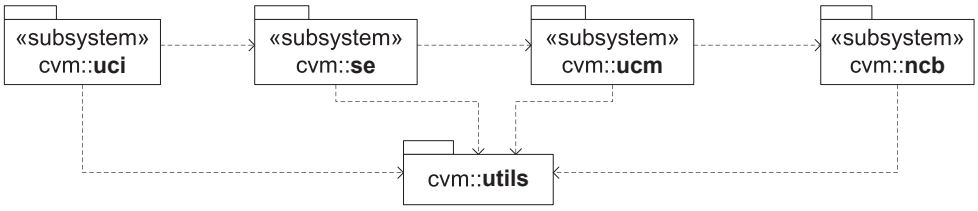

Figure 5.1: Top Level Architecture of the CVM prototype.

The original design of the CVM prototype consists of four major subsystems and a utility package as shown in Figure 5.1. These subsystems are the four components previously described in Section 2.1.3. A more in-depth conceptual description of

these components are presented by Deng et al. [17]. We present a summary of the components used in the prototype given below:

cvm::uci - the User Communication Interface (UCI) subsystem provides users will the ability to create, load, save and validate WF-CML models. The UCI also passes WF-CML models to SE for realization. Validation ensures that the models have appropriate values for all the required fields making them communication instances, for example each participant needs to have a valid user id. The main components of the UCI include (1) *Communications Modeling Environment* (CME), a graphical model environment, that provides expert users with the facility to create domain-specific communication models; (2) *User Interface* (UI), a user-friendly interface, that provides casual users with the facility to create communication models, both control and data instances, and (3) *Schema Transformation Environment* (STE) converts and validates models created in CME and UI into X-CML model instances before being passed onto SE. Figure 5.3 shows a diagram of the model created for the scenario described in Section 1.1. Appendix A9-A12 show the UI representation of the model for the control instance and two data instances. The original implementation of the *Communication modeling Environment* was developed using a combination of the Graphical Modeling Framework (GMF) [80], the Eclipse Modeling Framework (EMF)[66] and the Graphical Editing Framework [65]. 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
2. Use the class diagram in Step (1) to generate the Ecore model in EMF
3. Generate G-CML editor from EMF via a series of transformations
4. Create models using G-CML editor and output its XML representation

5. Transform the XML representation of G-CML model generated by G-CML editor into equivalent X-CML representation.

cvm::se - the functionality of SE is described in details in Section 4.1 and will not be revisited in its entirety at this point. Figure 5.2 shows a class diagram containing the main classes and packages used by SE. SE interfaces with UCI and UCM via the classes SE_Facade and UCM_Facade, respectively. The SE_Manager coordinates the activities of SE and the ConnectionProcessManager keeps track of the ConnectionProcess objects created per connection. The basic functionality of the SE_Manager (same as `SE_Controller` in Figure 4.1(b)) is described in Algorithm 6.1. The ConnectionProcess class coordinates the activities associated with each connection which includes the (re)negotiation and media transfer processes, shown in Figure 4.1. The classes NegotiationObject and MediaTransferObject define the state machines shown in the Tables A1 and A2, respectively. Schema analysis is performed in the utility package utils::handlers::schema. This functionality, described in algorithms analyze_CI (Algorithm 6.2) and analyze_DI (Algorithm 6.3) is, placed in the utils package since the UCI also needs to track changes to the control and data instances when updating the UI after receiving updated instances from SE.

cvm::ucm - UCM is designed to support the execution of communication control scripts, including system initialization, macro loading and interpretation, and exception and event handling and runtime media management. UCM handles events from the network communication broker (NCB) as well as internal events. Macros are loaded dynamically by the UCM manager which then delegates the the execution of the macros to the script interpreter. The runtime media management supports temporary storage of non-stream media, handling local user media request, and performing the actions associated with different form types. Wu et al. [81] describe the

UCM subsystem used in the current version of the prototype, providing additional details on how the functionality is implemented.

cvm::ncb - NCB exposes an API to the UCM that allows it to interact with the underlying communication frameworks/applications. The current version of the prototype interacts with Skype [60] through Java API Skype4Java [38]. NCB is developed using an autonomic architecture that will support other communication frameworks/applications and self-* capabilities. The prototype uses a bridge, NCBBridge, that interacts the Skype adapter, SkypeAdapter. Other adapters under development include Smack [41], JMML [45], and SIP Communicator [59]. The NCB is coordinated by the NCBManager that interacts with the communication services manager CommServicesManger that coordinates the activities of the various adapters that implement the NCBBridge. Allen et al. [1] provide additional details on the design and implementation of the NCB.

cvm::utils - this package provides support for the operations in other packages. The classes contained in cvm::util are used to analyze a schema since this functionality is required for both the UCI and SE. Schema analysis in the UCI is required when changes are made to the schema in SE and it is then sent to the UCI to be displayed in the UI. Chapter 4 describes the use of the schema analysis in SE. There is also an abstract class defined for event handlers that is used by all the layers in the CVM.

We extended the CVM platform to handle WF-CML models. The CVM still maintains the layered architecture as defined in Section 2.1.3. The major extensions were done in the synthesis engine (SE) and some changes were applied to the user communication interface (UCI). The two lower layers of CVM, User-centric Communication Middleware (UCM) and Network Communication Broker (NCB) were left unchanged. Prior to developing WF-CML, the communication modeling environment (CME) in the UCI was developed using Eclipse Modeling Framework (EMF)

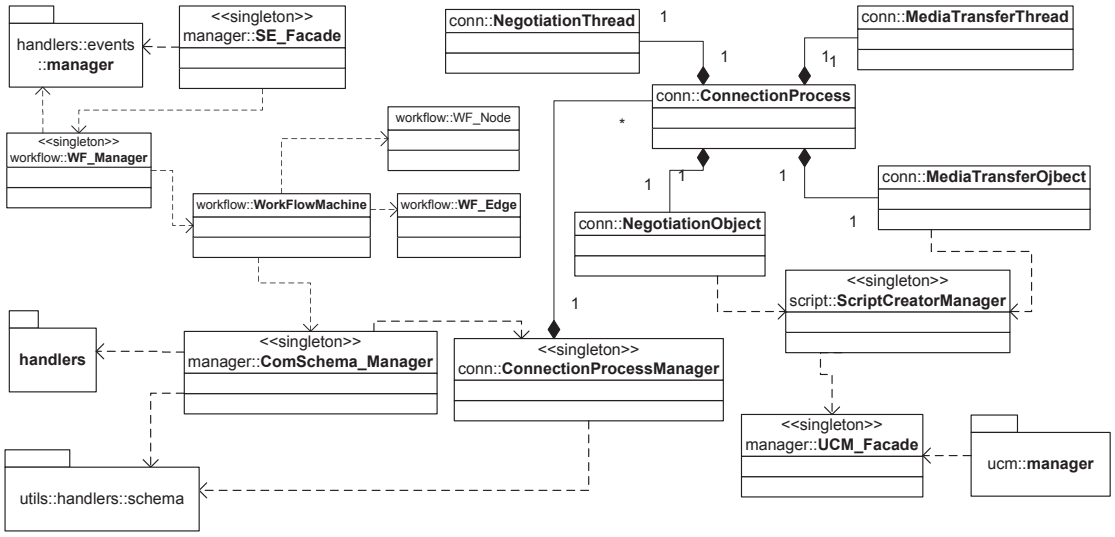

Figure 5.2: Class Diagram for the Synthesis Engine

[66]. However, since WF-CML models required nesting we decide to use the Visual Studio DSL Tools [14] to create the CME. Figure 5.3 shows a screen shot of the CME that the expert user uses to create the WF-CML model for the healthcare use case in Section 3.2.1. Figure 5.4 shows the user-friendly GUI that Dr. Burke uses to realize the WF-CML model.

SE was extended to handle the coordination of the communication services. Figure 5.2 shows the high-level design of the SE. Each executing WF-CML model has its thread of execution represented as *WorkFlowMachine*. Events are received from *handlers::events::manager* and managed by the *WF_Manager* and dispatched to the *WorkFlowMachine* during execution. Each *WorkFlowMachine* is responsible for traversing and maintaining its own WF-CML Hypergraph.

Table 5.1 shows some of the static metrics for the individual layers of the CVM. The first column contains the names of metrics, columns two through six contain the values of the metrics for the packages cvm::utils, cvm::uci, cvm::se, cvm::ucm and

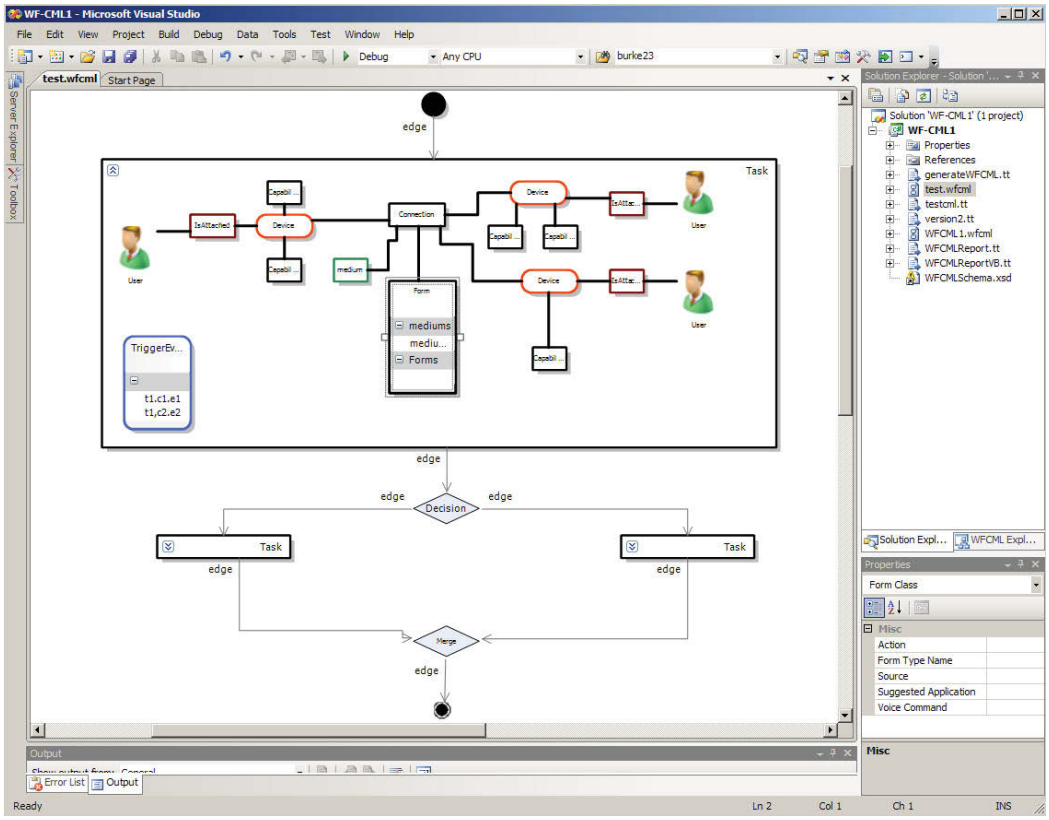

Figure 5.3: Modeling Environment for Coordinated UCCSs.

cvm::ncb, respectively. The last column on the left of the table contains the totals for the entire CVM. For example, the first row represents the single lines of code (SLOC), where package cvm::utils has 1,949 lines of code and the CVM has 10,416 lines of code. The metrics were obtained using Dependency Finder [64].

Table 5.1: Static metrics for the CVM prototype.

Metrics	*utils*	*uci*	*se*	*ucm*	*ncb*	*Totals*
SLOC	1,949	5,108	963	737	1,659	10,416
# packages	5	6	6	5	7	29
# classes	48	192	22	28	59	349
# methods	407	746	156	131	333	1,773

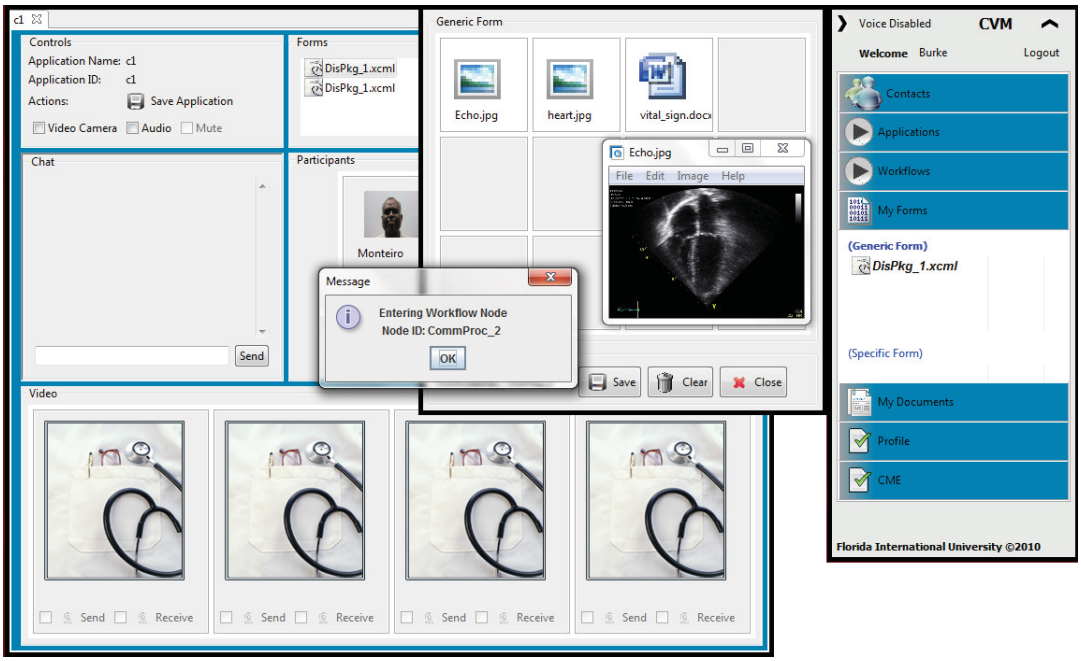

Figure 5.4: Dr. Burke's GUI showing (1) the message window requesting confirmation to advance in the workflow, and (2) Baby Jane's discharge package received from Dr. Monteiro.

5.2 Evaluating The Effort of Using WF-CML

In this section, we present a comparative study between WF-CML, Yet Another Workflow Language(YAWL)[72] and Microsoft Workflow Foundation (WF)[15] to show the advantages of using a domain-specific modeling language versus general purpose modeling language. We argue that by raising the level of abstraction, WFCML requires less development effort and expertise for modeling and realizing coordinated user-centric communication services.

YAWL is a workflow language based on a rigorous analysis of existing workflow management systems and workflow languages [72]. YAWL is supported by an extensible software system including an execution engine, a graphical editor, and a

worklist handler. The YAWL system is extensible by allowing external applications to interconnect with the workflow engine using a service-oriented approach.

Microsoft Workflow Foundations(WF) provides a programming model, in-process workflow engine and rehostable designer to implement long-running processes as workflows within .NET applications [15]. In WF, workflows are defined in XAML, but are usually edited using a graphical designer in Visual Studio. To execute the workflows, a WF Runtime is provided in the .NET Framework that includes common facilities for running and managing the workflows, providing feedback on execution progress, and hosting individual workflow instances.

5.2.1 Metrics

Based on our own experiences in designing and using DSML, as well as an extensive literature review of DSMLs, we identify and classify the effort associated with using DSMLs by breaking it down into categories. Figure 5.5 uses a feature diagram to present an initial pass at such a classification. The overall effort involves both the development effort for creating an application and runtime effort for executing the application. *Development Effort* is decomposed into Modeling Effort, Cognitive Effort and Scaffolding Effort. Modeling effort is the effort required to create the model, cognitive effort the effort to form the mental solutions to a problem, and the scaffolding effort the effort to complete the solution thereby making it executable. *Runtime Effort* is categorized into User Interaction Effort and System Execution Effort, denoting the required user interaction with the system, as well as the system resource utilization in realizing the application. Note that we do not claim this classification to be complete. Preliminary tasks such as learning effort, and binding effort (e.g. during code generation to customize coding components) that are technology and domain dependent are left out in this thesis.

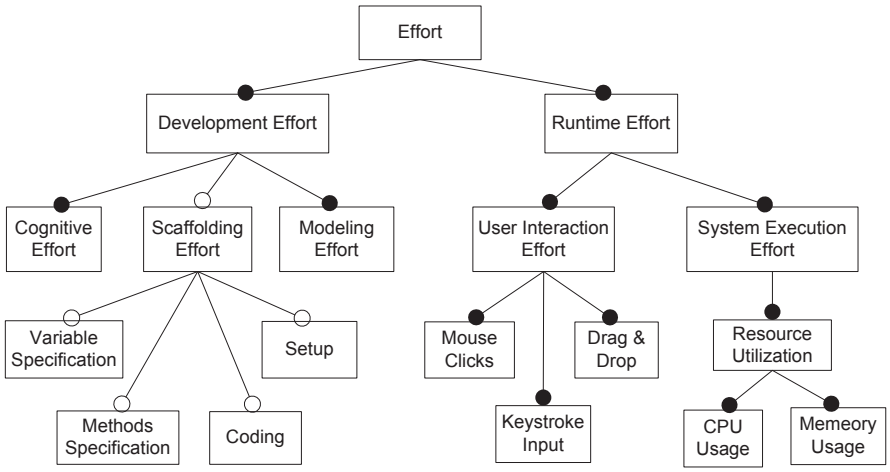

Figure 5.5: Feature Diagram to Classify Effort.

The details for the development effort are provided in this section using the structure provided in Figure 5.5. The development effort includes: modeling effort, cognitive effort and scaffolding effort.

Modeling Effort: In evaluating the modeling effort for developing DSML models we use the following graph-based metrics:

Size of model (SOM): defined as the "number of model elements" in a DSML model. It is analogous to SLOC used to represent program size. Various metrics have been proposed to measure the size of models, such as number of nodes, number of edges, number of attributes[8]. A generalized definition of SOM could be a weighted sum of each potential metric, with the weight validated by empirical studies:

$$SOM = \sum_{i=0}^{n} w_i \times m_i \tag{5.1}$$

where m_i denotes a single measure of the model size, like number of nodes or edges, and w_i represents its weight satisfying $\sum_{i=0}^{n} w_i$=1. A simple form of SOM is given

below where $\|N\|$ is the number of nodes:

$$SOM = \|N\| \tag{5.2}$$

Control Flow Complexity of Model (CFC): defined as the number of possible control flows in a DSML model (assuming it has explicit control structures). Cardoso [7] extended McCabe's cyclomatic number to measure the CFC of process models. He proposed that the CFC metric equates the number of decisions in the process flow. Every split in the model adds to the number of possible decisions as follows: CFC_{and} - AND-split adds 1; CFC_{xor} - XOR-split adds n; and CFC_{or} - OR-split adds $2^n - 1$. The CFC could be applied in the measurement of DSML models that have explicit control structures. The higher the CFC, the more complex the structure of the DSML model.

$$CFC(m) = \sum_{i \in xor} CFC_i + \sum_{j \in and} CFC_j + \sum_{k \in or} CFC_k \tag{5.3}$$

When computing the SOM and CFC for a given DSML model it may be necessary to flatten it if there are nested components.

Cognitive Effort: To determine the cognitive effort involved when developing DSML models, we use techniques from the work on software complexity metrics [57] and usability analysis of visual programing environments [31] .

Cognitive Weight(CW): measures the psychological complexity of a model in terms of the relative ease to understand and modify the model. Shao et al. [57] presents a metric to measure the difficulty or relative time and effort for comprehending a given piece of software modeled by a number of basic control structures (BCSs). The CWs for each BCS, based on empirical studies, are shown in Table 5.2. The CW of a software component (described as the component's cognitive functional size (CFS))

Table 5.2: Cognitive Weight for BCSs defined in [57]

Category	*Basic Control Structure*	*Cognitive Weight*
`Sequence`	Sequence	1
`Branch`	If-then-else	2
	Case	3
`Iteration`	For-do	3
	Repeat-until	3
	While-do	3
`Embedded Component`	Function Call	2
	Recursion	3
`Concurrency`	Parallel	4
	Interrupt	4

is defined as the sum of cognitive weights of its q linear blocks composed in individual BCSs, with each block consisting of m layers of nesting BCSs, and each layer with n linear BCSs.

$$CW(m) = \sum_{j=0}^{q} \prod_{k=0}^{m} \sum_{i=0}^{n} CW_{cs}(j,k,i) \tag{5.4}$$

In general to determine the CW of a DSML model we use Equation CW. Note that if the model does not have any nested structures then the CW is the sum of the CWs of all of its control structures. The higher the CW, the more difficulty it is to comprehend the model.

Closeness of Mapping Ratio (COMR): measures the effort needed to mentally construct a solution to the problem by translating the users' high-level goals into language primitives. Green [31] used the number of primitives and amount of syntax in different languages to infer the potential distance between the problem world and the program world. The more unusual primitives and lexical clutter, the more effort the developer has to put in arranging the components in a hard-to-remember structure with finicky syntax rules[31]. As an initial attempt we use COMR as the ratio of the number of problem-level language primitives (that are related to the user's inherent goals) divided by the number of solution-level primitives (that are "structural or semantic glue" and do not have counterparts in the problem domain). COMR

approximates the closeness of the mapping from the problem domain to the solution domain.

Scaffolding Effort: In evaluating the scaffolding effort in using DSMLs for creating applications, we introduced the following metrics:

Number of Additional LOC (NALOC): number of additional lines of code needed to generate a complete executable from a DSML model. It measures the additional coding effort required to realize applications using DMSLs.

Number of Additional Variables (NAV): number of additional variables be defined. In languages like YAWL [72] and BPEL [39], additional variables have to be defined that capture the data flow and storage needs of processes, contributing to the overall effort of adopting the DSML.

Number of Additional Methods (NAM): number of additional methods defined for completing the behavioral specification of the DSML model.

Number of Additional Components (NAC): number of external dependent software components that the developer has to manage or configure, such as additional database back-end support, servlet containers for hosting web services, required libraries, or DLLs that have to be imported or configured. It measures additional infrastructural support required for realizing the DSML model.

Runtime Effort: To determine the runtime effort for using DSML models at runtime, we consider two forms of effort: the user's effort in interacting with the DSML execution interface, and the system effort in terms of resource utilization. We proposed the following metrics:

Number of Mouse Clicks (NMC): number of mouse clicks to realize a user scenario.

Table 5.3: Metrics for CVM and YAWL Engine.

Static Metrics	*# Single Lines of Code*	*# of Classes*	*# of Methods*
`CVM`	11250	276	1522
`YAWL Engine`	43072	485	6738
Dynamic Metrics	*Avg. Memory Usage (Page File)*	*# Of Threads*	*Avg. Time for Executing WF Nodes(milliseconds)*
`CVM`	184	87	Audio Call – 874.2 Form Transfer – 944.4
`YAWL Engine`	374	161	Audio Call – 1950 Form Transfer – 1869.6

Number of Drag-and-Drops (NDD): number of drag and drop operations to realize a user scenario.

Number of Keystroke Inputs (NKI): number of keystroke inputs required from the user to realize a user scenario.

Memory Utilization (MU): amount of memory required by the underlying platform to realize the user scenario.

CPU Utilization (CPUU): the amount of CPU resources required by the underlying platform to realize the user scenario. We use the CPU time allocated to the related processes, and the thread count of the processes to infer the CPU utilization.

5.2.2 Design of Experiment

Experimental Set-up: The purpose of the experiment is to perform a comparative study that involves modeling and executing a scenario using WF-CML and two other DSMLs (YAWL and WF)and evaluate the effort involved in each case by using the metrics proposed in [83].

We modeled the same patient discharge scenario using their respective modeling environments in WF-CML (Figure 5.3), YAWL(Figure A20) and WF(Figure

A21)executed the models in their respective execution engines. YAWL requires the exact sequence of atomic communication tasks (calling the doctor followed by sending the discharge package, followed by inviting another doctor into the call, etc.) to be specified at design time. Also it requires explicit data flows to be specified through the mapping of input/output parameters to task variables. In WF-CML, basic nodes of communication processes are modeled with declarative CML models, which specify high level communication needs as opposed to detailed steps of communication. Finally in WF, the specific sequences of communication activities also need to be explicitly defined and the custom activities implemented for invoking communication services required. The generic steps for the experimental setup are as follows:

- Set up CVM and YAWL engine on Skype as communication service provider; implement custom activities in WF to perform communication-specific tasks
- Specify the same collaborative communication process using WF-CML, YAWL and Microsoft WF model in their respective graphical editors or designers
- Load and execute service specifications in respective execution platforms

The detailed procedure of the experiment is shown in Table 5.4. The table is divided into two rows, the first row describes the system setup, and the second row the steps to develop the executable for the DSML model.

Data Collection: Collecting the development effort data involved obtaining the values for SOM, CFC, and CW which were straightforward. The values for COMR required the classification of language primitives into problem-level and solution-level constructs. Measuring the user effort required counting the number of mouse clicks, drag-and-drop operations and user inputs during the execution of the scenario. The data for system effort was obtained by: (1) instrumenting the code with time stamps

to measure the elapsed execution time for the workflow engine to interpret the workflow specification, and (2) using the task manager to obtain the number of threads allocated to the workflow process. We use the number of page files allocated for the execution engine as an approximate indication of memory utilization.

Table 5.4 shows the comparison of YAWL, WF and WF-CML in terms of setup prerequisites, development effort (including the specification of the high level process flow and the specification of the service for each task node), required expertise as well as ease-of-change. Table 5.3 shows the collected static metrics of the YAWL implementation and CVM implementation, including number of lines of node, number of classes and methods. Dynamic metrics for the two implementations are also shown in Table 5.3, including the number of threads and the memory usage needed to start up the CVM and YAWL engine, and the average execution time for realizing certain workflow nodes. We discuss the results of the experiments in the next subsection.

Table 5.4: Experimental Setup and Procedure

	YAWL	*WF*	*WF-CML*
`System Setup`	1.YAWL system (editor, engine) 2.Servlet Container (Apache Tomcat) 3.Database back-end (PostgreSQL) 4. Skype4Java (Skype API for Java)	1.NET Framework 2.Skype4COM (Skype API for C#)	1.CVM system 2.Skype4Java (Skype API for Java)
`Development Process`	1.Create models in YAWL editor 2.Develop and deploy web services that invoke Skype API calls 3.Register deployed service in YAWL engine 4.Bind communication tasks in the YAWL specification to registered web services 5.Define task variables and net variables and mapping between input/output parameters to these variables	1.Create models in WF Designer 2.Implement customized workflow activities that realize Skype actions 3.Develop the workflow client to interact with the workflow engine and Skype by hosting the workflow and connecting to the Skype proxy	1.Create models in WF-CML editor 2.Specify communication services nodes using CML 3.Specify trigger events for advancing communication nodes
`Required Expertise`	1. Developing/deploying customized web services in IDE 2. Understanding of web service bindings using parameter mappings 3. Understanding of data flows and predicate specification	Basic understanding of Microsoft Windows Foundation	Basic understanding of WF-CML
`Efforts For Changing Needs`	1. For changes in YAWL process, update model in YAWL editor 2. For changes in the service of activity nodes, repeat four steps outlined above	repeat four steps outlined above	1. For changes in WF-CML process, update model in WF-CML editor 2. For changes in comm. service nodes, update CML model dynamically

5.2.3 Experimental Results

Table 5.5: Development Effort

Modeling/ Cognitive	*SOM Top Level/Total*	*CFC*	*CW*	*COMR*
YAWL	21/21	9	61	14/17
WF	77/77	20	22	13/64
WF-CML	7/54	2	3	7/4
Scaffolding	*NALOC*	*NAV*	*NAM*	*NAC*
YAWL	857	34	3	38
WF	1265	77	1	58
WF-CML	0	0	0	0

Models for the scenario using the three DSMLs, WF-CML, YAWL, and WF were created during the study. We show the screen shots of these models in the appendices. The collected metrics for the three techniques are shown in Tables 5.5 and 5.6. Table 5.5 presents the comparison of YAWL, WF and WF-CML in terms of manual, cognitive and scaffolding effort. We used the designed metrics to measure the manual effort in terms of size of model (SOM), to measure the cognitive effort in terms control flow complexity (CFC), cognitive weight (CW) and closeness of mapping ratio (COMR), and finally to measure the scaffolding effort in terms of number of additional lines of code (NALOC), number of additional variables (NAV), number of methods (NAM) and number of additional components (NAC).

Table 5.6 illustrates the runtime metrics in terms of user effort and system execution effort. To measure the user effort, we collected the number of mouse clicks (NMC), number of drag and drop operations (NDD) as well as the number of keyboard inputs (NKI). The system execution effort, which mainly involved with the utilization of system resources during the runtime execution of DSML models, is measured using the memory utilization (number of page files) and CPU utilization (number of thread counts and number of milliseconds).

Table 5.6: Runtime Effort

User Effort	*NMC*	*NDD*	*NKI*
YAWL	3	0	9
WF	15	0	1
WF-CML	3	5	1
System Effort	*MU (Page File)*	*CPUU*	
		Threads	*(milliseconds)*
YAWL	374	161	1909.8
WF	128	49	303
WF-CML	184	87	909.3

5.2.4 Discussions

The results of our experiments provide evidence on potential productivity gains of WF-CML over general purpose workflow languages like YAWL and WF: WF-CML leads to reduced development effort during the service creation process and less user execution effort during runtime. In addition, by avoiding full-blown workflow solutions, it could achieve a lightweight resource utilization comparing with general purpose workflow languages such as YAWL, but might result in additional memory and CPU usage compared to a low-level languages such as WF. We discuss the experimental results in more details below:

Development Effort: Table 5.4 and Table 5.5 illustrate the effort involved in creating coordinated communication services. In using YAWL, modelers need to develop, deploy and register web services (known as YAWL services) with the YAWL engine, define task and net variables, use parameter mapping to specify data flow, and define predicate guards for conditional branching. In using Windows WF, developers need to develop a workflow host application that interacts with Windows WF through the WorkflowRuntime class, implement customized workflow activities/services using C#programing language, and populate the WorkflowRuntime class with implemented

activities for use during the workflow execution. Using WF-CML, the modeler is only required to specify the communication services using CML model and define trigger events. These development tasks, classified as scaffolding effort, illustrates the amount of effort developers have to go through to produce complete workflow solutions. Moreover, the complexities of the created models, which is measured in terms of the modeling effort and cognitive effort also provides insights

System Runtime Efficiency: Table 5.3 provides evidence on the lightweight nature of the CVM platform over the YAWL engine and its supporting systems e.g., Apache Tomcat and PostgreSQL. Since YAWL offers complete workflow solutions, it has the most resource utilization due to its heavy weight workflow engine. Also, YAWL supports the interaction between running workflow instances with external applications exposed as services, which also contributes to a more heavyweight workflow solution. Due to our restriction to coordinated UCCSs, we trade generality for a lightweight workflow approach, demonstrated by the decrease in the size () as well runtime overhead of the system. In addition, the additional level of indirection (caused by the increased level of abstraction) might cause extra utilization of system resources, as demonstrated by more memory and CPU usage of WF-CML compared to WF. This illustrate a trade off between ease-of-use and runtime system resource utilization.

Runtime effort: Table 5.6 provides evidence on the reduced user execution effort in interpreting these models at runtime. While using WF-CML, users are required to use more drag-and-drop operations, but less number of mouse clicks and keyboard inputs. Assume that drag and drop operation, mouse click operations and keyboard input operations have the same level of ease, we could appropriate the overall effort by summing up three different types of user execution effort. The results will be leaning forwards the WF-CML requiring less overall effort during the service realization.

Conclusion: We argue that by raising the level of abstraction, WF-CML requires less development effort and expertise for modeling and realizing coordinated user-centric communication services. In addition, the restricted domain of WF-CML results in a more lightweight execution engine (the CVM) than a general purpose workflow engine.

Limitations on the experiment: The prototype is a proof-of-concept implementation of the research ideas proposed. Therefore, evaluations using the current prototype must be subject to implementation and performance constraints that might leads to not-optimal experimental results. Also, the evaluation of "expressive and efficiency" is a hard problem, and require extensive user studies, which is not feasible due to the length of the program. This leads to potential biased comparison of the user-studies due to the lack of a more comprehensive set of user participants.

On the other hand, since our work is just an initial attempt towards quantitative measurement of effort in using DSMLs during application development. There needs to be more empirical studies to validate the metrics presented in the paper. Also there are several limitations with our study as described below: (1) Only three DSMLs are investigated in this study. There needs to be a more comprehensive review of different categories of DSMLs to consolidate the classification of effort presented in the paper. (2) Metrics like CFC only measure a class of DSMLs that have explicit flow structures, such as process modeling languages (e.g., BPEL [39]) and the Call Processing language[51]. For declarative DSMLs, CFC would not be appropriate to measure the structural complexity. (3) The measurement of the cognitive effort in using DSMLs lack empirical evidence. More empirical evaluation is required to determine the impact of the cognitive effort in creating DSML models. The long term vision of this research includes the estimation of a single effort value for each DSML by doing a weighted sum of each of the potential metrics.

Threats to validity: The threats to our experimentation can be classified under two major headings 1) threats to internal validity, and 2) threats to external validity. Threads to internal validity includes: (1) the occurrence of unexpected network delays and events in actual communication channels; (2) the response delay at various participant end-points, and (3) the instrumentation of the synthesis engine using TPTP as a single measuring instrument appears not to be suitable for the experiments. Using Eclipse TPTP to capture the metrics during execution of the SE resulted in the time for each iteration for a scenario being much longer than anticipated. In addition, the machine on which the experiments were performed had to be restarted several times during the experiments. To mitigate the threats to internal validity, we repeated each scenario ten times, removed the maximum and minimum values and computed the average of the reaming 8 values.

Threats to external validity include: the generalization of the chosen networks, demonstration computers and specific communication frameworks that CVM uses e.g., Skype [60]. To avoid threats to external validity, we chose different machines (both desktops and laptops), in a combination of wired and wireless local area networks. However, the current prototype implementation relies on Skype for actual delivery of communication services, which diminishes the generality of the approach on multiple communication frameworks. Skype does not identify the machine it uses to mix the audio in a conference call and this may have impacted the results for realization times. Part of our future work is to integrate other frameworks such as Smack[41], Asterisk [20] and MSN Messenger [45] into the NCB. We are also working on an approach for dynamic adaptation that would move a conference call from Skype to Asterisk when the numbers of participants in a call reaches a particular threshold. Asterisk is a software implementation of a telephone private branch exchange (PBX) that supports VoIP services.

5.3 Chapter Summary

In this chapter, we presented implementation details of the Communication Virtual Machine prototype to demonstrate the practicality of this approach. We also designed and performed comparative studies with other workflow languages and measured the effort of creating communication applications using different approaches. The studies provide arguments for the productivity gains of using WF-CML.

CHAPTER 6

CONCLUSION

In this chapter, we conclude this dissertation by presenting a summary of the research as well as future directions. The first section summarizes the major contributions of this dissertation. Then, we list several future directions of this work.

6.1 Contribution Summary

This dissertation applies a domain-specific modeling approach to the specification and realization of coordinated user-centric communication services. The major contribution is: the design of a domain specific modeling language tailored for coordinating user-centric communication services. This language facilitates the specification of coordinated user-centric communication services in a way that is driven by end users and yet enables rapid realization. The specific objectives that have been achieved in this dissertation include:

- The syntactic design of a domain specific modeling language for coordinating user-centric communication services, including a domain analysis and a metamodel definition that is comprised of an abstract syntax and static semantics.
- The definition of dynamic semantics for WF-CML models that includes detailed semantic specifications and a conceptual design of an execution architecture for the dynamic synthesis of WF-CML models.
- The development of a Communication Virtual Machine (CVM) prototype for the rapid realization of coordinated communication services and experimental studies to demonstrate the benefits of this modeling technique.

Next, we list the future directions of this dissertation work.

6.2 Future Work

In this section, we present the future work of this dissertation research in multiple directions. First, we will further consolidate the semantic specifications of WF-CML for more complicated scenarios involving distributed workflow execution. The current semantics of WF-CML are limited to the coordination of communication services locally. It does not support service coordination logic that migrates among participants. We plan to revisit the dynamic semantics of WF-CML to support distributed service coordination. A WF-CML model may include a global view of the service coordination logic in a communication process. As the WF-CML model is passed on to different participants, each participant will extract his/her local view (a subset of the model that is of his concern), either for efficiency or security reasons. Depending on the policies specified in the model, each party will be limited to access or modify a particular part of the model. This will also raise interesting questions on how to keep consistency among multiple local views of the model. We plan to investigate further in this direction.

Second, we plan to generalize this approach into a generic approach for describing interpreted domain-specific modeling languages (i-DSMLs). Although this dissertation focus on the user-centric communication domain, it provides significant insights into how user-driven modeling approaches can be used to support the development of domain-specific services. Defining the semantics for WF-CML to support the direct interpretation of models (without first generating code) raises the level of abstraction potentially addressing the essential problems of software complexity and visibility. Especially we are looking at definition dynamic semantics for i-DSMLs

through model comparison and infer actions based on changes in models. i-DSMLs provide a simple graphical modeling language for domain experts(e.g. in healthcare, scientific collaboration or disaster management,microgrid) to quickly create domain-specific application. In fact, some PhD students in the research group are currently working on developing a DSML for the MicroGrid domain [49, 47] to monitor loads, activate and deactivate energy consumers and producers, using the same methodologies introduced in this dissertation. The initial results show that the concepts used in CML and CVM, including the semantics specification approach, translate easily to the MicroGrid domain.

Last but not least, we plan to extend the current comparative studies to include more control groups. For instance, we will incorporate BPEL [39], UML activity diagrams, and analyze how these languages could be used to model and realize the patient discharge scenario. The ultimate goal is to perform user studies in a realistic setting and get their feedbacks about the usability of this language. Their feedbacks will be used to further improve the language features. We plan to conduct surveys amongst developers that use WF-CML while developing applications for coordinating user-centric communication services. Motivated by [37], we will use the known success factors of the use of DSMLs, such as improved maintainability and ease of re-use, and assert how well WF-CML scores on all of them. The analysis of the results of such case studies will also provide insights on which conditions should be fulfilled in order to increase the chances of success in using a DSML in a real life case.

BIBLIOGRAPHY

[1] Andrew A. Allen. *Abstractions to Support Dynamic Adaptation of Communication Frameworks for User-Centric Communication*. PhD thesis, Florida International University, April 2011.

[2] Apple Inc. Facetime for iphone 4, October 2010. http://www.apple.com/iphone.

[3] Apple Inc. ipad, 2010. http://www.apple.com/ipad/.

[4] Stefan Arbanowski, Sven Van Der Meer, and Stephan Steglich. User-centric communications. In *Proceedings of the 8th IEEE International Conference on Telecommunications (ICT 2001)*, pages 425–444, 2001.

[5] Egon Börger and Bernhard Thalheim. Modeling workflows, interaction patterns, web services and business processes: The asm-based approach. In *ABZ '08: Proceedings of the 1st international conference on Abstract State Machines, B and Z*, pages 24–38, Berlin, Heidelberg, 2008. Springer-Verlag.

[6] R. P. Burke and J. A. White. Internet rounds: A congenital heart surgeon's web log. *Seminars in Thoracic and Cardiovascular Surgery*, 16(3):283–292, 2004.

[7] Jorge Cardoso. How to measure the control-flow complexity of web processes and workflows. In *The Workflow Handbook*, pages 199–212, 2005.

[8] Michel Chaudron and Christian F. Lange. Second international workshop on model size metrics. In *Models in Software Engineering: Workshops and Symposia at MoDELS 2007*, pages 89–92, Berlin, Heidelberg, 2008. Springer-Verlag.

[9] Kyo C.Kang, Sholom G. Cohen, james A. Hess, William E. Novak, and A. Spencer Peterson. Feature-oriented domain analysis. Technical Report CMU/SEI-90-TR-21, CMU, Nov 1990.

[10] Tony Clark, Andy Evans, Paul Sammut, and James Willans. *Applied Metamodelling, A Foundation for Language Driven Development*. 2004.

[11] Peter J. Clarke, Vagelis Hristidis, Yingbo Wang, Nagarajan Prabakar, and Yi Deng. A declarative approach for specifying user-centric communication. In *Proceeding of the International Symposium on Collaborative Technologies and Systems (CTS 2006)*, pages 89 – 98. IEEE, May 2006.

[12] Workflow Management Coalition. The workflow reference model, 2008. http://www.wfmc.org.

[13] Louise K. Comfort, Kilkon Ko, and Adam Zagorecki. Coordination in rapidly evolving disaster response systems. *American Behavioral Scientist*, 48(3):295–313, 2004.

[14] Steve Cook, Gareth Jones, Stuart Kent, and Alan Wills. *Domain-specific development with visual studio dsl tools*. Addison-Wesley Professional, 2007.

[15] Microsoft Corporation. Windows workflow foundation, 2010. http://msdn.microsoft.com/en-us/netframework/aa663328.aspx.

[16] Cyberbridges. Center for Internet Augmented Research and Assessment. http://www.cyberbridges.net/archive/summary.htm (Nov2007).

[17] Yi Deng, S. Masoud Sadjadi, Peter J. Clarke, Vagelis Hristidis, Raju Rangaswami, and Yingbo Wang. CVM - a communication virtual machine. *Journal of Systems and Software*, 2008. (in press).

[18] Krishna Kishore Dhara, Tim I. Ross, and Venkatesh Krishnaswamy. A framework for developing user-centric services for communication end-points. In *Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE*, pages 1 –6, 30 2009-dec. 4 2009.

[19] Davide Di Ruscio, Frédéric Jouault, Ivan Kurtev, Jean Bézivin, and Alfonso Pierantonio. Extending amma for supporting dynamic semantics specifications of dsls. Technical Report n. 06.02, Laboratoire d'Informatique de Nantes-Atlantique (LINA), 2006.

[20] Digium. Asterisk, June 2010. http://www.asterisk.org/.

[21] Marlon Dumas and Arthur ter Hofstede. Uml activity diagrams as a workflow specification language. 2185:76–90, 2001. 10.1007/3-540-45441-1_7.

[22] Tracy Yee Emily Carrier and Rachel A. Holtzwart. Coordination between emergency and primary care physicians, national institute for health care reform. http://www.nihcr.org/ED-Coordination.html#section2.

[23] Naeem Esfahani, Sam Malek, ao P. Sousa, Jo Hassan Gomaa, and Daniel A. Menascé. A modeling language for activity-oriented composition of service-oriented software systems. In *MODELS '09*, pages 591–605, Berlin, Heidelberg, 2009. Springer-Verlag.

[24] Rik Eshuis and Roel Wieringa. An execution algorithm for uml activity graphs. In *Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools*, pages 47–61, London, UK, 2001. Springer-Verlag.

[25] Rik Eshuis and Roel Wieringa. A formal semantics for uml activity diagrams - formalising workflow models. Number TR-CTIT-01-04 in CTIT technical reports series, Enschede, 2001. University of Twente, Centre for Telematics and Information Technology. http://doc.utwente.nl/37504/.

[26] FEMA. DisasterHelp. http://www.disasterhelp.gov/start.shtm (May 2008).

[27] Robert France and Bernhard Rumpe. Model-driven development of complex software: A research roadmap. In *FOSE '07: 2007 Future of Software Engineering*, pages 37–54, Washington, DC, USA, 2007. IEEE Computer Society.

[28] Patrick Freudenstein, Jan Buck, Martin Nussbaumer, and Martin Gaedke. Model-driven construction of workflow-based web applications with domain-specific languages. In *Proceedings of the 3rd International Workshop on Model-Driven Web Engineering MDWE 2007, Como, Italy, July 17, 2007*, volume 261 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

[29] Google. Android developers, May 2010. http://developer.android.com.

[30] Google Project Hosting. bpmn2bpel: A tool for translating bpmn models into bpel processes, June 2011. http://code.google.com/p/bpmn2bpel/.

[31] T. R. G. Green and M. Petre. Usability analysis of visual programming environments: a 'cognitive dimensions' framework. *Journal of Visual Languages and Computing*, 7:131–174, 1996.

[32] Object Management Group. Omg unified modeling language (omg uml), infrastructure, v2.1.2. Technical report, November 2007.

[33] Object Management Group. Technical report, May 2011. http://www.omg.org/technology/documents/modeling_spec_catalog.htm#QVT.

[34] Object Management Group. Omg model driven architecture. Technical report, May 2011. http://www.omg.org/mda/.

[35] The Parlay Group. Parlay/osa specifications, August 2010. http://etsi.org/WebSite/Technologies/OSA.aspx.

[36] Zef Hemel, Ruben Verhaaf, and Eelco Visser. Webworkflow: An object-oriented workflow modeling language for web applications. In *MoDELS '08: Proceedings of the 11th international conference on Model Driven Engineering Languages and Systems*, pages 113–127, Berlin, Heidelberg, 2008. Springer-Verlag.

[37] Felienne Hermans, Martin Pinzger, and Arie Deursen. Domain-specific languages in practice: A user study on the success factors. In *MODELS '09*, pages 423–437, Berlin, Heidelberg, 2009. Springer-Verlag.

[38] Koji Hisano and Bart Lamot. Skype4Java. https://developer.skype.com/wiki/Java_API [10 March 2010].

[39] IBM. *BPEL4WS, Business Process Execution Language for Web Services Version 1.1*, 2003.

[40] IBM. The power of unifying communications and collaboration. Unified communications and collaboration White paper, November 2009.

[41] Ignite Realtime. Smack api 3.1.0, Jan. 2009. http://www.igniterealtime.org/.

[42] Internet2. Internet2 working groups, and special interest groups. http://www.internet2.edu/working-groups.html (Sept 2006).

[43] Jack Jachner, Emmanuel Darmois, Scott Petrack, and Tim Ozugur. Rich presence: A new user communications experience. Alcatel Telecommunications Review, 2005. http://www1.alcatel-lucent.com/doctypes/articlepaperlibrary/pdf/ATR2005Q1/T0503-Rich_presence-EN.pdf.

[44] JML Development Team. The java media framework api, March 2010. http://www.oracle.com/technetwork/java/javase/tech/index-jsp-140239.html.

[45] JML Development Team. Java msn messenger library, March 2010. http://sourceforge.net/projects/java-jml/.

[46] David Krebs. The mobile software stack for voice, data, and converged handheld devices, April 2005. Mobile and Wireless Practice Venture Development Corporation.

[47] J. D. Kueck, R.H. Staunton, S. D. Labinov, and B.J. Kirby. Microgrid energy management system, Jan 2003. http://www.ornl.gov/sci/btc/apps/Restructuring/ORNLTM2002242rev.pdf (Dec. 2010).

[48] Philippe Lasserre and Dennis Kan. User-centric interactions beyond communications. Alcatel Telecommunications Review, March 2007.

[49] Michael R. Lavelle. Micro-grid applications, January 2010. http://www.lavelleenergy.com/index.php/micro-grids/72-micro-grid-applications.

[50] MORFEO open source community. Ezweb project, 2010. http://ezweb.tid.es.

[51] Columbia University Network Working Group. Call Processing Language (CPL), 2004. http://www.ietf.org/rfc/rfc3880.txt.

[52] Hanne R. Nielson and Flemming Nielson. *Semantics with applications: a formal introduction*. John Wiley & Sons, Inc., New York, NY, USA, 1992.

[53] J. L. Peterson. Petri net theory and modeling of systems. *Journal of Networks*, 1981.

[54] Gordon D. Plotkin. A structural approach to operational semantics. *The Journal of Logic and Algebraic Programming*, 60-61:17–139, 2004.

[55] Davide Sangiorgi and David Walker. *PI-Calculus: A Theory of Mobile Processes*. Cambridge University Press, New York, NY, USA, 2001.

[56] Ismael H. F. Santos, Martin Göbel, Alberto B. Raposo, and Marcelo Gattass. A multimedia workflow-based collaborative engineering environment for oil gas industry. In *VRCAI '04: Proceedings of the 2004 ACM SIGGRAPH international conference on Virtual Reality continuum and its applications in industry*, pages 112–119, New York, NY, USA, 2004. ACM.

[57] Jingqiu Shao and Yingxu Wang. A new measure of software complexity based on cognitive weights. *IEEE Canadian Journal of Electrical and Computer Engineering*, 83:69–74, 2003.

[58] Youngmee Shin, Chorong Yu, Seunghwa Chung, and Sangki Kim. End-user driven service creation for converged service of telecom and internet. *Advanced International Conference on Telecommunications*, 0:71–76, 2008.

[59] SIP Communicator Development Team. SIP Communicator. http://sip-communicator.org/ [14 March 2010].

[60] Skype Limited. Skype developer zone, Feb. 2010. https://developer.skype.com/.

[61] Thomas Stahl, Markus Voter, Jorn Bettin, Arno Haase, Simon Helsen, and Krzysztof Czarnecki. *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons, first edition, 2003.

[62] Sun Microsystems and OpenCloud. Jain-slee, 2008. http://www.jainslee.org/.

[63] Sasu Tarkoma, Bharat Bhushan, Erno Kovacs, Herma van Kranenburg, Erwin Postmann, Robert Seidl, and Anna V. Zhdanova. Spice: A service platform for future mobile ims services. In *World of Wireless, Mobile and Multimedia Networks, 2007. WoWMoM 2007. IEEE International Symposium on a*, pages 1 –8, June 2007.

[64] Jean Tessier. Dependency Finder. http://depfind.sourceforge.net/ [21 March 2010].

[65] The Eclipse Foundation. Graphical editing framework (gef). http://en.wikipedia.org/wiki/Eclipse_Modeling_Framework (March 2007).

[66] The Eclipse Foundation. Eclipse modeling framework project (emf), 2011. http://www.eclipse.org/modeling/emf/.

[67] The Eclipse Foundation. Eclipse software development kit, 2011. http://www.eclipse.org/.

[68] The JAIN-SIP Project. Java api for sip signaling, March 2010. https://jain-sip.dev.java.net/.

[69] Triskell Team. Kermeta - breathe life into your metamodels, July 2010. http://www.kermeta.org/.

[70] NY User Centric Communications New York. User centric communications, 2009. http://www.usercentric.net/.

[71] Wil M. P. van der Aalst. The application of petri nets to workflow management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.

[72] W.M.P. van der Aalst and A. H. M. Ter Hofstede. Yawl: Yet another workflow language. *Information Systems*, 30:245–275, 2003.

[73] Arie van Deursen, Paul Klint, and Joost Visser. Domain-specific languages: an annotated bibliography. *ACM SIGPLAN Notices*, 35(6):26–36, 2000.

[74] Markus Völter and Thomas Stahl. *Model-Driven Software Development : Technology, Engineering, Management*. John Wiley & Sons, June 2006.

[75] Yingbo Wang. *Modeling Communication Services for Rapid Realization*. PhD thesis, Florida International University, August 2009.

[76] Yingbo Wang, Peter J. Clarke, Yali Wu, Andrew Allen, and Yi Deng. Runtime models to support user-centric communication. Models@runtime Workshop in conjunction Models 2008. http://www.comp.lancs.ac.uk/ bencomo/MRT/ (Jan. 2009).

[77] Yingbo Wang, Yali Wu, A. Allen, B. Espinoza, P.J. Clarke, and Yi Deng. Towards the operational semantics of user-centric communication models. In *Computer Software and Applications Conference, 2009. COMPSAC '09. 33rd Annual IEEE International*, volume 1, pages 254 –262, july 2009.

[78] M. Weber, S. Hock, G. Partsch, A. Scheller-Huoy, G. Schneider, and J. Schweitzer. Combining multimedia collaboration and workflow management. pages 114–119, Sep 1997.

[79] S.A. White. Introduction to bpmn. Technical report, 2004. http://www.bpmn.org/Documents/Introduction20to20BPMN.pdf.

[80] Wikipedia. Graphical modeling framework. http://en.wikipedia.org/wiki/Graphical_Modeling_Framework (March 2007).

[81] Yali Wu, Andrew Allan, Yingbo Wang, Frank Hernandez, Peter J. Clarke, and Yi Deng. A user-centric communication middleware for cvm. 2008.

[82] Yali Wu, Andrew A. Allen, Frank Hernandez, Robert B. France, and Peter J. Clarke. A domain-specific modeling approach to realizing user-centric communication. *Journal of Software Practice and Experience (SP&E)*, 2011. (to appear).

[83] Yali Wu, Frank Hernandez, Francisco Ortega, Peter J. Clarke, and Robert France. Measuring the effort for creating and using domain-specific models. In *Proceedings of 10th DSM Workshop*, 2010.

[84] Juan C. Yelmo, Jose M. del Alamo, Ruben Trapero, and Yod-Samuel Martin. A user-centric approach to service creation and delivery over next generation networks. *Computer Communications*, In Press, Corrected Proof:–, 2010.

Appendix

APPENDICES

WF-CML For Patient Discharge Scenario

Graphical WF-CML For Validation in Kermeta

Kermeta Code For Checking WF-CML Static Semantics

Screenshots for Realizing the Patient Discharge Scenario

Execution Trace in Kermeta Simulation - 1

Execution Trace in Kermeta Simulation - 2

Complete Static Semantics for WF-CML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<commWorkFlow>
   <initial id="i1"><outgoingEdgeRef id="x1"/></initial>
     <commProcNode id="CommProc_2" procName="ConferencingForPatientDischargeWithSC">
       <cmlSchema>
             <controlSchema communicationID="">
               <connection bandwidth="1111" connectionID="c1">
                 <device isLocal="false" isVirtual="true" deviceID="d1">
                   <deviceCapability>TextFile</deviceCapability>
                   <deviceCapability>LiveAudio</deviceCapability>
                   <deviceCapability>VideoFile</deviceCapability>
                   <deviceCapability>NonStreamFile</deviceCapability>
                 </device>
                 <device isLocal="false" isVirtual="true" deviceID="d2">
                   <deviceCapability>TextFile</deviceCapability>
                   <deviceCapability>VideoFile</deviceCapability>
                   <deviceCapability>NonStreamFile</deviceCapability>
                   <deviceCapability>LiveAudio</deviceCapability>
                 </device>
                 <device isLocal="true" isVirtual="false" deviceID="d3">
                   <deviceCapability>TextFile</deviceCapability>
                   <deviceCapability>LiveAudio</deviceCapability>
                   <deviceCapability>NonStreamFile</deviceCapability>
                   <deviceCapability>VideoFile</deviceCapability>
                 </device>
                 <mediumTypeNameRef>audio streaming</mediumTypeNameRef>
                 <formTypeNameRef>patient record</formTypeNameRef>
              </connection>
              <mediumType voiceCommand="start audio" derivedFromBuiltInType="LiveAudio"
                          mediumTypeName="audio streaming"/>
              <formType action="send" suggestedApplication="default" formTypeName="PatientDischarge">
                    <mediumDataType>TextFile</mediumDataType>
                    <mediumDataType>NonStreamFile</mediumDataType>
                    <mediumDataType>VideoFile</mediumDataType>
              </formType>
              <person personRole="Discharge Physician" personID="burke23" personName="Dr. Burke"/>
              <person personRole="Senior Clinician" personID="monteiro45" personName="Dr. Monteiro"/>
              <isAttached deviceID="d1" personID="burke23"/>
              <isAttached deviceID="d2" personID="monteiro45"/>
       </controlSchema>
       <data connectionID="c1" communicationID="">
          <medium mediumDataType="LIVE_AUDIO" mediumName="LiveAudio" mediumURL="LiveAudio"
          state="on"/></data>
       <data connectionID="c1" communicationID="">
           <form formDataType="Generic" formID="DisPkg_1" suggestedApplication="browser"
                    voiceCommand="" action="send">
              <medium mediumDataType="TextFile" mediumName="RecSum-Jane.txt" mediumURL=
                                  "U:\burke23\files\RecSum-Jane.txt" mediumSize="" lastModified=""
                                  validityPeriod=""  firstTransferTime="" voiceCommand="" action=""/>
              <medium mediumDataType="NonStreamFile" mediumName="xRay-Jane.jpg" mediumURL=
                                  "U:\burke23\files\xRay-Jane.jpg" mediumSize="" lastModified=""
                                  validityPeriod="" firstTransferTime="" voiceCommand="" action=""/>
```

Figure A1: XML Version of WF-CML For Modeling Healthcare Use Case - 1

```
    <medium mediumDataType="VideoFile" mediumName="HeartEcho-Jane.mpg" mediumURL=
                                        "U:\burke23\files\HeartEcho-Jane.mpg" mediumSize="" lastModified=""
                                       validityPeriod="" firstTransferTime="" voiceCommand="" action=""
  </form></data>
 </cmlSchema>
 <triggerEvent eventID = "t1">
        <commProcEvent eventID = "t1.c1">
           <atomicEvent eventID = "t1.c1.e1" type="FormEvent" formTypeName = "PatientDischarge"
                       formName = "DisPkg_1" statusFT = "Received" temporalOp = "Before"
                       temporalValue= "24" temporalStart = "Sent">
           <DomainPropertyPredicate key = "validity" relOp="equalTo"  value="true"/></atomicEvent>
        </commProcEvent>
        <commProcEvent eventID = "t1.c2">
           <atomicEvent eventID = "t1.c2.e2" type="FormEvent" startTimer = "Sent"
                statusFT = "NotReceived" formTypeName = "PatientDischarge" formName = "DisPkg_1"
                       temporalStatus = "after" temporalValue= "24" temporalStart = "Sent"/>
        </commProcEvent>
 </triggerEvent>
 <outgoingEdgeRef id="x2"/>
 <incomingEdgeRef id="x1"/>
</commProcNode>
<decision id="d1">
      <outgoingEdgeRef id="x3"/>
      <outgoingEdgeRef id="x4"/>
      <incomingEdgeRef id="x2"/>
</decision>
<commProcNode  id="CommProc_2" procName="Conferencing with NP and AP">
      <cmlSchema>
         <controlSchema communicationID="">
           <connection bandwidth="1111" connectionID="c2">
              <device isLocal="false" isVirtual="true" deviceID="d1">
                 <deviceCapability>TextFile</deviceCapability>
                 <deviceCapability>LiveAudio</deviceCapability>
                 <deviceCapability>VideoFile</deviceCapability>
                 <deviceCapability>NonStreamFile</deviceCapability>
              </device>
              <device isLocal="false" isVirtual="true" deviceID="d2">
                 <deviceCapability>TextFile</deviceCapability>
                 <deviceCapability>VideoFile</deviceCapability>
                 <deviceCapability>NonStreamFile</deviceCapability>
                 <deviceCapability>LiveAudio</deviceCapability>
              </device>
              <device isLocal="true" isVirtual="false" deviceID="d3">
                 <deviceCapability>TextFile</deviceCapability>
                 <deviceCapability>LiveAudio</deviceCapability>
                 <deviceCapability>NonStreamFile</deviceCapability>
                 <deviceCapability>VideoFile</deviceCapability>
              </device>
              <mediumTypeNameRef>audio streaming</mediumTypeNameRef>
              <formTypeNameRef>patient record</formTypeNameRef>
           </connection>
```

Figure A2: XML Version of WF-CML For Modeling Healthcare Use Case - 2

```
        <formType action="send" suggestedApplication="default" formTypeName="PatientDischarge">
          <mediumDataType>TextFile</mediumDataType>
          <mediumDataType>NonStreamFile</mediumDataType>
          <mediumDataType>VideoFile</mediumDataType>
        </formType>
        <person personRole="Discharge Physician" personID="burke23" personName = "Dr. Burke" />
        <person personRole="Nurse Practitioner" personID="p2" personName = "Nurse Smith"/>
        <person personRole="Attending Physician" personID="p3" personName = "Dr. Wang"/>
        <isAttached deviceID="d1" personID="p1"/>
        <isAttached deviceID="d2" personID="p2"/>
        <isAttached deviceID="d3" personID="p3"/>
      </controlSchema>
      <data connectionID="c2" communicationID="">
       <form formDataType="Generic" formID="DisPkg_1" suggestedApplication="browser"
                    voiceCommand="" action="send">
            <medium mediumDataType="TextFile" mediumName="RecSum-Jane.txt" mediumURL="U:\
                    burke23\files\RecSum-Jane.txt" mediumSize="" lastModified="" validityPeriod=""
                     firstTransferTime="" voiceCommand="" action=""/>
            <medium mediumDataType="NonStreamFile" mediumName="xRay-Jane.jpg" mediumURL=
                     "U:\burke23\files\xRay-Jane.jpg" mediumSize="" lastModified="" validityPeriod=""
                    firstTransferTime="" voiceCommand="" action=""/>
            <medium mediumDataType="VideoFile" mediumName="HeartEcho-Jane.mpg" mediumURL=
                     "U:\burke23\files\HeartEcho-Jane.mpg" mediumSize="" lastModified=""
                     validityPeriod="" firstTransferTime="" voiceCommand="" action=""/>
                    </form>
                    </data>
      </cmlSchema>
      <triggerEvent eventID = "t2">
            <commProcEvent eventID = "t2.c1">
               <atomicEvent eventID = "t2.c1.e1" type="FormEvent" statusMT = "Sent"
                    formTypeName = "PatientDischarge" formName = "DisPkg_1"/>
            </commProcEvent>
      </triggerEvent>
      <outgoingEdgeRef id="x5"/>
      <incomingEdgeRef id="x3"/>
 </commProcNode>
<commProcNode  id="comm3" procName=" C with Interim Package">
    <cmlSchema>
            <controlSchema communicationID="">
        <connection bandwidth="1111" connectionID="c3">
          <device isLocal="false" isVirtual="true" deviceID="d1">
            <deviceCapability>TextFile</deviceCapability>
            <deviceCapability>LiveAudio</deviceCapability>
            <deviceCapability>VideoFile</deviceCapability>
            <deviceCapability>NonStreamFile</deviceCapability>
          </device>
          <device isLocal="false" isVirtual="true" deviceID="d2">
            <deviceCapability>TextFile</deviceCapability>
            <deviceCapability>VideoFile</deviceCapability>
            <deviceCapability>NonStreamFile</deviceCapability>
            <deviceCapability>LiveAudio</deviceCapability>
          </device>
```

Figure A3: XML Version of WF-CML For Modeling Healthcare Use Case - 3

```
            <device isLocal="true" isVirtual="false" deviceID="d3">
                  <deviceCapability>TextFile</deviceCapability>
                  <deviceCapability>LiveAudio</deviceCapability>
                  <deviceCapability>NonStreamFile</deviceCapability>
                  <deviceCapability>VideoFile</deviceCapability>
            </device>
            <mediumTypeNameRef>audio streaming</mediumTypeNameRef>
            <formTypeNameRef>patient record</formTypeNameRef>
       </connection>
       <mediumType voiceCommand="send File" derivedFromBuiltInType="TextFile"
            mediumTypeName="InterimNote"/>
       <person personRole="Attending Physician" personID="wang12" personName = "Dr. Wang"/>
       <isAttached deviceID="d1" personID="wang12"/>
   </controlSchema>
   <data connectionID="c3" communicationID="">
          <medium mediumDataType="TextFile" mediumName="InterimNote-Jane.txt" mediumURL=
                        "U:\burke23\files\InterimNote-Jane.txt" mediumSize="" lastModified=""
                validityPeriod="" firstTransferTime="" voiceCommand="" action=""/>
   </data>
  </cmlSchema>
  <triggerEvent eventID = "t3">
    <commProcEvent eventID = "t3.c1">
        <atomicEvent eventID = "t3.c1.e1" type="FormEvent" statusMT = "Sent"
              mediumTypeName = "InterimNote" mediumName="InterimNote-Jane"/>
     </commProcEvent>
  </triggerEvent>
  <outgoingEdgeRef id="x6"/>
  <incomingEdgeRef id="x4"/>
</commProcNode>
<merge id="m1">
       <outgoingEdgeRef id="x7"/>
       <incomingEdgeRef id="x5"/>
       <incomingEdgeRef id="x6"/>
   </merge>
   <final id="f1">
       <incomingEdgeRef id="x7"/>
   </final>
   <Edge type="RegularEdge" source="i1" target="comm1" id="x1"/>
   <Edge source="comm1" target="d1" id="x2"/>
   <Edge type="DecisionEdge" source="d1" target="comm2" id="x3">
        <EdgeAnnotation type = "guardIf"><atomicEvent eventID= "t1.c1.e1"/></EdgeAnnotation>
   </Edge>
   <Edge type="DecisionEdge" source="d1" target="comm3" id="x4">
         <EdgeAnnotation type = "guardElse"/>
    </Edge>
   <Edge type="RegularEdge" source="comm2" target="m1" id="x5"/>
   <Edge type="RegularEdge" source="comm3" target="m1" id="x6"/>
   <Edge type="RegularEdge" source="m1" target="f1" id="x7"/>

</commWorkFlow>
```

Figure A4: XML Version of WF-CML For Modeling Healthcare Use Case - 4

```
<?xml version="1.0" encoding="UTF-8"?>
<cmlWorkFlow:CommWorkFlow xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:cmlWorkFlow="
        platform:/resource/CML/metamodels/CML.ecore#//cmlWorkFlow">
<activities xsi:type="cmlWorkFlow:Initial" id="0">
        <outgoingEdges xsi:type="cmlWorkFlow:RegularEdge" source="0" target="1" id="0.1"/>
</activities>
<activities xsi:type="cmlWorkFlow:CommActivity" id="1" activityName="activity1">
   <outgoingEdges xsi:type="cmlWorkFlow:RegularEdge" source="1" target="d1" id="1.d1"/>
   <cmlSchema/>
</activities>
<activities xsi:type="cmlWorkFlow:Decision" incomingEdges="1.d1" id="d1">
   <outgoingEdges xsi:type="cmlWorkFlow:DecisionEdge" source="d1" target="2" id="d1.2">
     <guard expression=""/>
   </outgoingEdges>
   <outgoingEdges xsi:type="cmlWorkFlow:DecisionEdge" source="d1" target="3" id="d1.3"/>
</activities>
<activities xsi:type="cmlWorkFlow:CommActivity" incomingEdges="d1.2" id="2" activityName="activity2">
        <outgoingEdges xsi:type="cmlWorkFlow:RegularEdge" source="2" target="f" id="2.m1"/>
</activities>
<activities xsi:type="cmlWorkFlow:CommActivity" incomingEdges="d1.3" id="3" activityName="activity3">
   <outgoingEdges xsi:type="cmlWorkFlow:RegularEdge" source="3" target="m1" id="3.m1"/>
</activities>
<activities xsi:type="cmlWorkFlow:Merge" incomingEdges="2.m1 3.m1" id="m1">
   <outgoingEdges xsi:type="cmlWorkFlow:RegularEdge" source="m1" target="f" id="m1.f"/>
</activities>
<activities xsi:type="cmlWorkFlow:Final" incomingEdges="3.m1" id="f"/>
</cmlWorkFlow:CommWorkFlow>
```

Figure A5: WF-CML Model in EMF for Kermeta Validation - 1

```
<?xml version="1.0" encoding="UTF-8"?>
<cmlWorkFlow:CommWorkFlow xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://
        www.w3.org/2001/XMLSchema-instance" xmlns:cmlWorkFlow="platform:/resource/CML/
        metamodels/CML.ecore#//cmlWorkFlow" xmlns:cmlWorkflowEvent="platform:/resource/CML/
        metamodels/CML.ecore#//cmlWorkFlowEvent">
  <activities xsi:type="cmlWorkFlow:Initial" id="0">
        <outgoingEdges xsi:type="cmlWorkFlow:RegularEdge" source="0" target="1" id="0.1"/>
  </activities>
  <activities xsi:type="cmlWorkFlow:CommActivity" id="1" owner="" activityName="activity1">
        <outgoingEdges xsi:type="cmlWorkFlow:RegularEdge" source="1" target="d1" id="1.d1"/>
        <cmlSchema/>
        <triggerEvent xsi:type="cmlWorkflowEvent:MediaReceived" eventID="event1" connID=""
        dataType="" dataID=""/>
  </activities>
  <activities xsi:type="cmlWorkFlow:Decision" incomingEdges="1.d1" id="d1">
        <outgoingEdges xsi:type="cmlWorkFlow:DecisionEdge" source="d1" target="2" id="d1.2">
                <guard expression="event1.form.isValidated"/>
        </outgoingEdges>
        <outgoingEdges xsi:type="cmlWorkFlow:DecisionEdge" source="d1" target="3" id="d1.3">
                <guard expression="else"/>
        </outgoingEdges>
  </activities>
  <activities xsi:type="cmlWorkFlow:CommActivity" incomingEdges="d1.2" id="2"
        activityName="activity2">
        <outgoingEdges xsi:type="cmlWorkFlow:RegularEdge" source="2" target="f" id="2.m1"/>
        <cmlSchema/>
        <triggerEvent xsi:type="cmlWorkflowEvent:MediaSent" eventID="event2"/>
  </activities>
  <activities xsi:type="cmlWorkFlow:CommActivity" incomingEdges="d1.3" id="3"
activityName="activity3">
        <outgoingEdges xsi:type="cmlWorkFlow:RegularEdge" source="3" target="m1" id="3.m1"/>
        <cmlSchema/>
        <triggerEvent xsi:type="cmlWorkflowEvent:MediaSent" eventID="event3"/>
  </activities>
  <activities xsi:type="cmlWorkFlow:Merge" incomingEdges="2.m1 3.m1" id="m1">
        <outgoingEdges xsi:type="cmlWorkFlow:RegularEdge" source="m1" target="f" id="m1.f"/>
  </activities>
  <activities xsi:type="cmlWorkFlow:Final" incomingEdges="3.m1" id="f"/>
</cmlWorkFlow:CommWorkFlow>
```

Figure A6: A WF-CML Model in EMF for Kermeta Validation - 2

```
@mainClass "cml::InvariantChecker"
@mainOperation "main"
package cml;
require kermeta
require "platform:/resource/CML/metamodels/CML.ecore"
require "platform:/resource/CML/utilities/SchemaHelper.kmt"
require "platform:/resource/CML/constraints/CMLStaticSemantics.kmt"

using utilities
using cml::cmlControlSyntax
using cml::cmlWorkFlow
using kermeta::standard
using kermeta::persistence
using kermeta::exceptions


class InvariantChecker
{
      operation main() : Void is do
         checkModel("cml/test3.cmlcontrolsyntax")
            // checkModel ("/wf/test2.cmlworkflow")
      end

      operation checkModel (prg: String) is do
        var rep : EMFRepository init EMFRepository.new
        var theSchema : controlSchema init SchemaHelper.new.loadEMFAutomaton(rep,
            "platform:/resource/CML/test_models/"+prg,"platform:/resource/CML/metamodels/
            CML.ecore")

        stdio.writeln("Check WFR : start")
        check_CML_WFR(theSchema)
        // check_CML_WFR(thewf)
        stdio.writeln("Check WFR : end")

        rescue (err : ConstraintViolatedInv )
            stdio.writeln("exception generated in Schema ")
            stdio.writeln(err.toString())
            if (err.message == void) then
                stdio.writeln("Abstract syntax violated")
            else
                stdio.writeln(err.message)
            end
        end

        operation check_CML_WFR(theSchema : controlSchema) is do
            theSchema.partyLocal.checkAllInvariants()
            rescue (err : ConstraintViolatedInv)
                stdio.writeln("~~~~~~~~~~~~~~~~~~~~~~~~~")
                stdio.writeln("exception generated in Local Party ")
                stdio.writeln(err.toString)
                if (err.message == void) then
                        stdio.writeln("Abstract syntax violated in Local Party")
                else
                        stdio.writeln(err.message)
                end
        end
```

Figure A7: Kermeta Code For Checking Static Semantics of WF-CML - 1

```
theSchema.connection.each{ s |
      do
            s.checkInvariants()
            // Call the invariant verification
            rescue (err : ConstraintViolatedInv)
                  stdio.writeln("exception generated in Connection ")
                  stdio.writeln(err.toString)
                  if (err.message == void) then
                        stdio.writeln("Abstract syntax violated")
                  else
                        stdio.writeln(err.message)
                  end
      end}

theSchema.connection.each{ s |
      do
            s.dataType.checkAllInvariants()
            // Call the invariant verification
            rescue (err : ConstraintViolatedInv)
                  stdio.writeln("exception generated in dataTypeAttached ")
                  stdio.writeln(err.toString)
                  if (err.message == void) then
                        stdio.writeln("Abstract syntax violated in dataTypeAttached")
                  else
                        stdio.writeln(err.message)
                  end
      end}

theSchema.connection.each{ s |
      do
            s.partyRemote.each{r |
                  do
                        r.checkAllInvariants()
                        // Call the invariant verification
                        rescue (err : ConstraintViolatedInv)
                              stdio.writeln("exception generated in Connection ")
                              stdio.writeln(err.toString)
                              if (err.message == void) then
                                    stdio.writeln("Abstract syntax violated in Remote Party")
                              else
                                     stdio.writeln(err.message)
                              end
                  end}
      end}

theSchema.checkInvariants()
rescue (err : ConstraintViolatedInv)
      stdio.writeln("exception generated in Schema ")
      stdio.writeln(err.toString)
      if (err.message == void) then
            stdio.writeln("Abstract syntax violated in Schema")
      else
            stdio.writeln(err.message)
      end
end

operation check_WF_WFR (theWF: CommWorkFlow) is do
      theWF.checkAllInvariants()
end }
```

Figure A8: Kermeta Code For Checking Static Semantics of WF-CML - 2

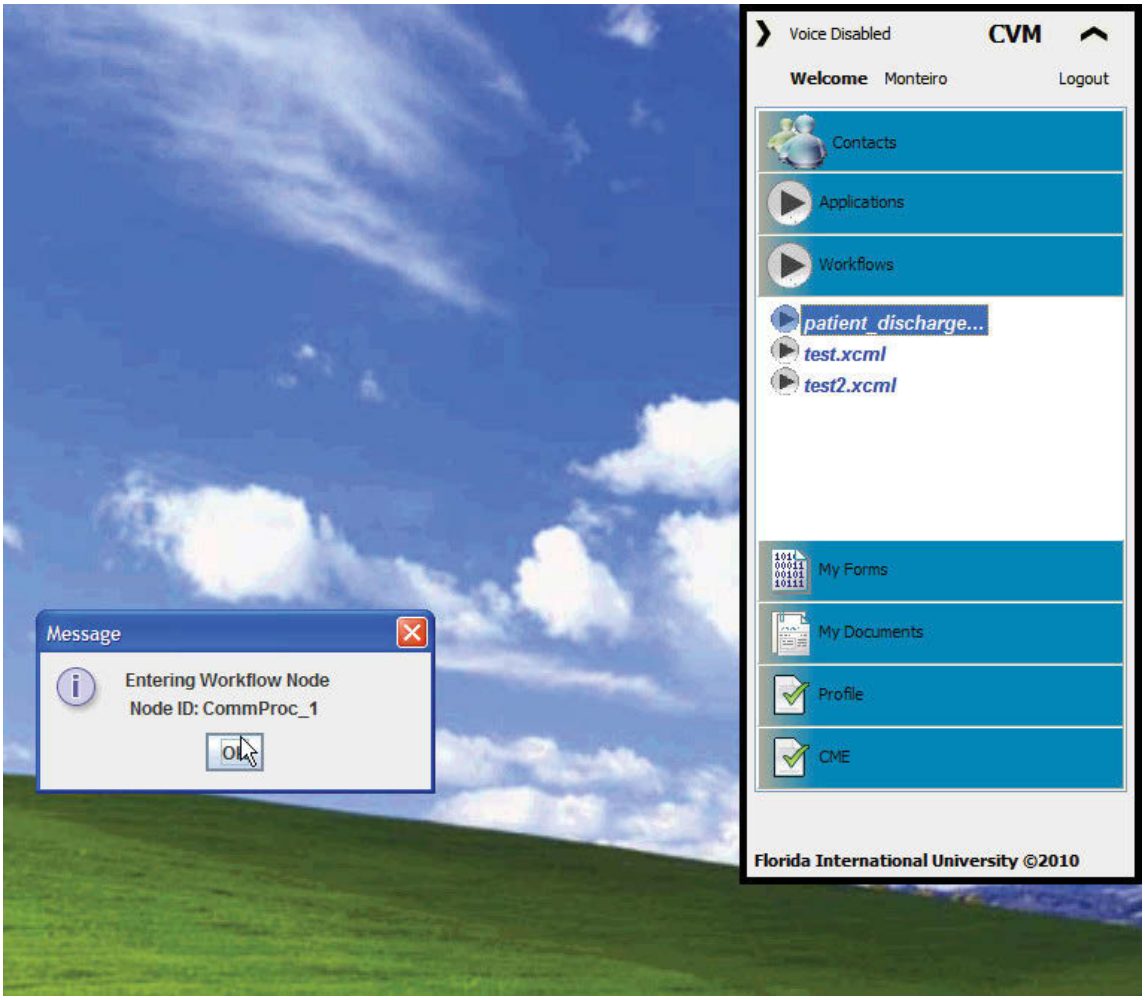

Figure A9: User Interface For CVM In Realizing Scenario - First step

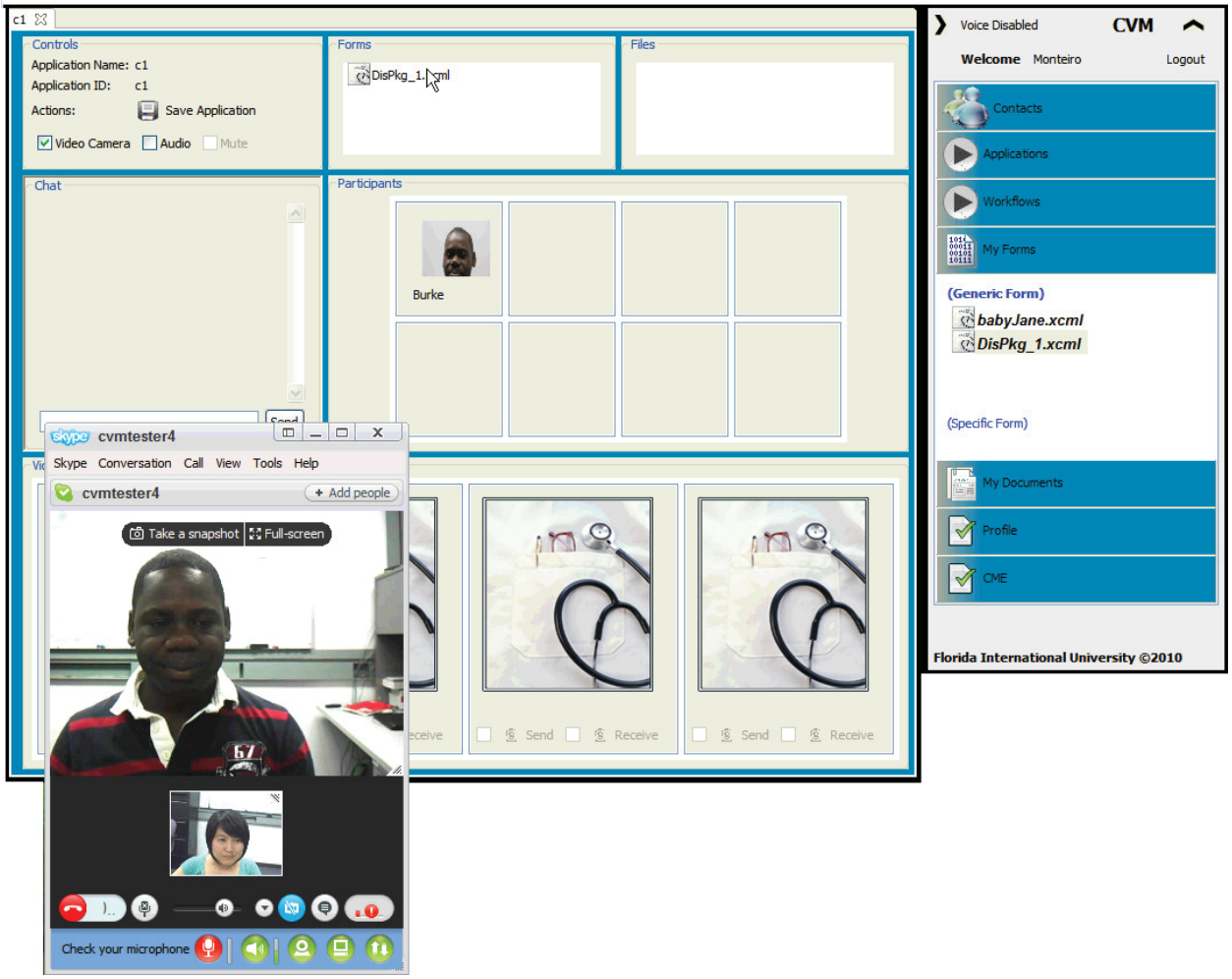

Figure A10: User Interface For CVM In Realizing Scenario - Second step

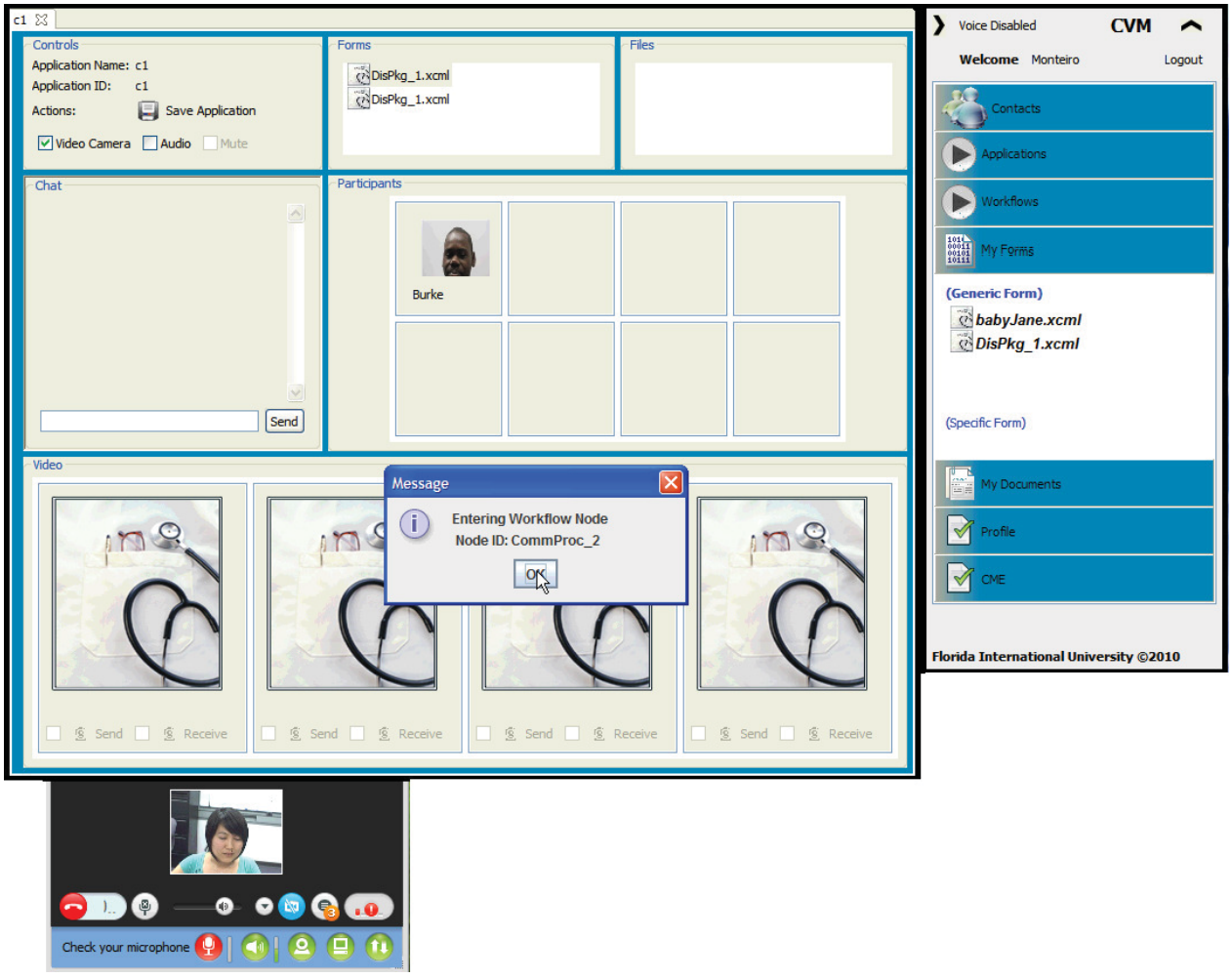

Figure A11: User Interface For CVM In Realizing Scenario - Third step

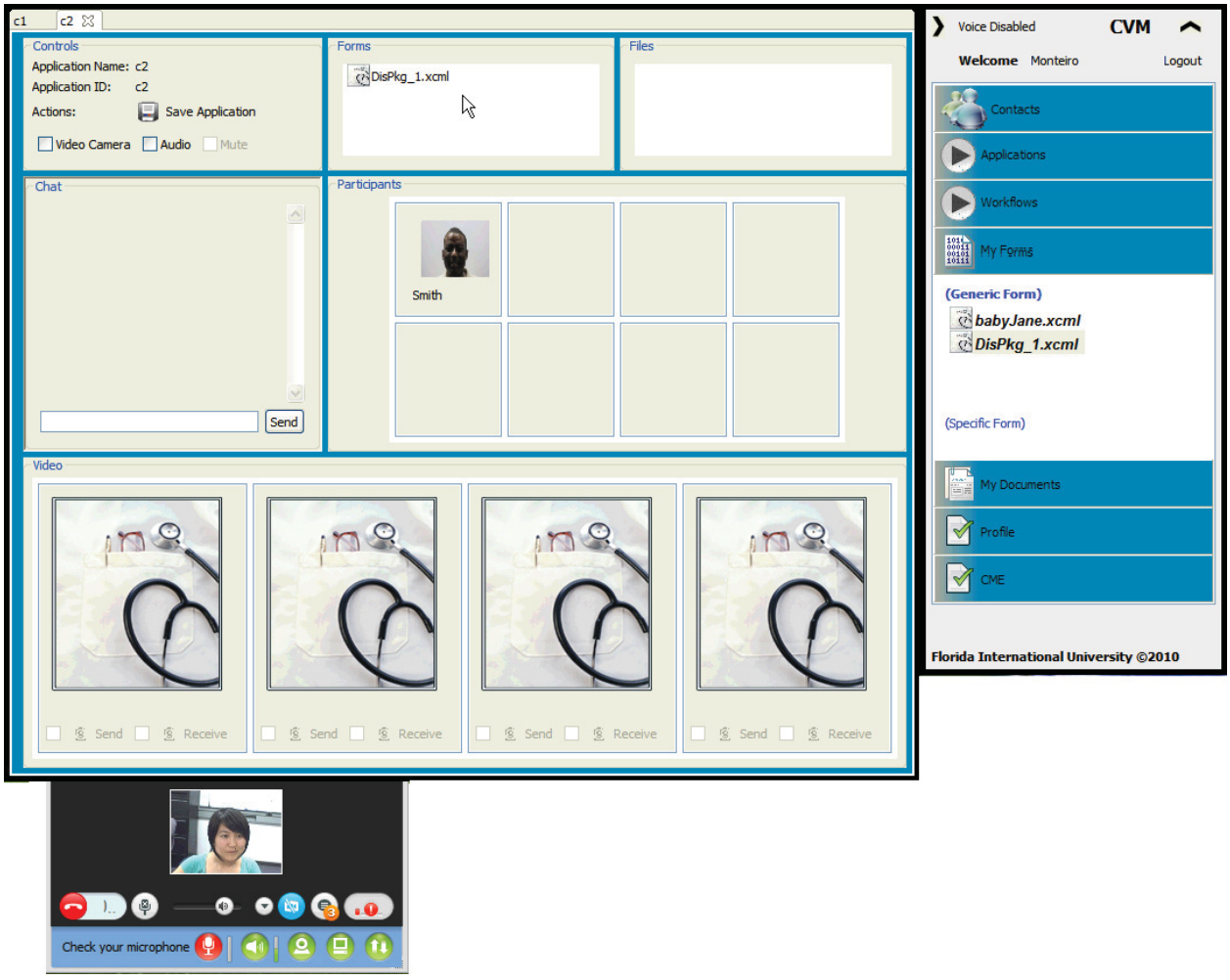

Figure A12: User Interface For CVM In Realizing Scenario - Fourth step

```
WF Controller started..
Will subsequent events be input seperately or all in one file?no
Enter filename: --> C:\Users\Yali\workspace\kermeta_CVM\CML_Seman\schemas\wf\list8.txt
AH execution started ...
Progress: Activity call A
Activity call A is executing...
Condition of Action is: true
Script Command produced :
createConnections("connection1")
sendSchema ("connection1" , "yali1028" , "Clarkep,Aalle004" , "test1.cmlcontrolsyntax" , "")
Current state of negotiation : WaitingSameCS
Schema contents
*********************************
Local Party : Yali textFile LiveAudio AudioFile
Connection :
        Remote Party: Peter
        Remote Party: Andrew
        Supported Form Type : PatientRecord
                Sub Medium Type : summary
                Sub Medium Type : echo
        Supported Medium Type : audio streaming
        Supported Medium Type : text file

*********************************
Evaluating the next step
Number of events in queue 7
Event ID is CommActivitySchema_control/test1_r_aalle004.cmlcontrolsyntax
Not relevant for current workflow state, but could belong to a future state.
Still in Activity call A
Condition of Action is: true
Script Command produced :
Current state of negotiation : WaitingSameCS
Schema contents

*********************************
Local Party : Yali textFile LiveAudio AudioFile
Connection :
        Remote Party: Peter
        Remote Party: Andrew
        Supported Form Type : PatientRecord
                Sub Medium Type : summary
                Sub Medium Type : echo
        Supported Medium Type : audio streaming
        Supported Medium Type : text file

*********************************
Is more events queued false
Event ID is CommActivitySchema_control/test1_r_clarkep.cmlcontrolsyntax
Not relevant for current workflow state, but could belong to a future state.
Still in Activity call A
Condition of Action is: true
The updated schema is sent to UCI
Script Command produced :
sendSchema ("connection1" , "yali1028" , "Clarkep,Aalle004" , "test1.cmlcontrolsyntax" , "")
Current state of negotiation : NegComplete
Schema contents
*********************************
```

Figure A13: Kermeta Simulation Trace in Realizing Sunny Day Scenario - 1

```
WF Controller started..
Will subsequent events be input seperately or all in one file?no
Enter filename: --> C:\Users\Yali\workspace\kermeta_CVM\CML_Seman\schemas\wf\list8.txt
AH execution started ...
Progress: Activity call A
Activity call A is executing...
Condition of Action is: true
Script Command produced :
createConnections("connection1")
sendSchema ("connection1" , "yali1028" , "Clarkep,Aalle004" , "test1.cmlcontrolsyntax" , "")
Current state of negotiation : WaitingSameCS
Schema contents
*******************************
Local Party : Yali textFile LiveAudio AudioFile
Connection :
        Remote Party: Peter
        Remote Party: Andrew
        Supported Form Type : PatientRecord
                Sub Medium Type : summary
                Sub Medium Type : echo
        Supported Medium Type : audio streaming
        Supported Medium Type : text file

*******************************
Evaluating the next step
Number of events in queue 7
Event ID is CommActivitySchema_control/test1_r_aalle004.cmlcontrolsyntax
Not relevant for current workflow state, but could belong to a future state.
Still in Activity call A
Condition of Action is: true
Script Command produced :
Current state of negotiation : WaitingSameCS
Schema contents

*******************************
Local Party : Yali textFile LiveAudio AudioFile
Connection :
        Remote Party: Peter
        Remote Party: Andrew
        Supported Form Type : PatientRecord
                Sub Medium Type : summary
                Sub Medium Type : echo
        Supported Medium Type : audio streaming
        Supported Medium Type : text file

*******************************
Is more events queued false
Event ID is CommActivitySchema_control/test1_r_clarkep.cmlcontrolsyntax
Not relevant for current workflow state, but could belong to a future state.
Still in Activity call A
Condition of Action is: true
The updated schema is sent to UCI
Script Command produced :
sendSchema ("connection1" , "yali1028" , "Clarkep,Aalle004" , "test1.cmlcontrolsyntax" , "")
Current state of negotiation : NegComplete
Schema contents
*******************************
```

Figure A14: Kermeta Simulation Trace in Realizing Sunny Day Scenario - 2

```
WF Controller started..
Will subsequent events be input seperately or all in one file?no
Enter filename: --> C:\Users\Yali\workspace\kermeta_CVM\CML_Seman\schemas\wf\list8.txt
AH execution started ...
Progress: Activity call A
Activity call A is executing...
Condition of Action is: true
Script Command produced :
createConnections("connection1")
sendSchema ("connection1" , "yali1028" , "Clarkep,Aalle004" , "test1.cmlcontrolsyntax" , "")
Current state of negotiation : WaitingSameCS
Schema contents
*******************************
Local Party : Yali textFile LiveAudio AudioFile
Connection :
        Remote Party: Peter
        Remote Party: Andrew
        Supported Form Type : PatientRecord
                Sub Medium Type : summary
                Sub Medium Type : echo
        Supported Medium Type : audio streaming
        Supported Medium Type : text file

*******************************
Evaluating the next step
Number of events in queue 7
Event ID is CommActivitySchema_control/test1_r_aalle004.cmlcontrolsyntax
Not relevant for current workflow state, but could belong to a future state.
Still in Activity call A
Condition of Action is: true
Script Command produced :
Current state of negotiation : WaitingSameCS
Schema contents

*******************************
Local Party : Yali textFile LiveAudio AudioFile
Connection :
        Remote Party: Peter
        Remote Party: Andrew
        Supported Form Type : PatientRecord
                Sub Medium Type : summary
                Sub Medium Type : echo
        Supported Medium Type : audio streaming
        Supported Medium Type : text file

*******************************
Is more events queued false
Event ID is CommActivitySchema_control/test1_r_clarkep.cmlcontrolsyntax
Not relevant for current workflow state, but could belong to a future state.
Still in Activity call A
Condition of Action is: true
The updated schema is sent to UCI
Script Command produced :
sendSchema ("connection1" , "yali1028" , "Clarkep,Aalle004" , "test1.cmlcontrolsyntax" , "")
Current state of negotiation : NegComplete
Schema contents
*******************************
```

Figure A15: Kermeta Simulation Trace in Realizing Sunny Day Scenario - 3

```
WF Controller started..
Will subsequent events be input seperately or all in one file?no
Enter filename: --> C:\Users\Yali\workspace\kermeta_CVM\CML_Seman\schemas\wf\list8.txt
AH execution started ...
Progress: Activity call A
Activity call A is executing...
Condition of Action is: true
Script Command produced :
createConnections("connection1")
sendSchema ("connection1" , "yali1028" , "Clarkep,Aalle004" , "test1.cmlcontrolsyntax" , "")
Current state of negotiation : WaitingSameCS
Schema contents
*********************************
Local Party : Yali textFile LiveAudio AudioFile
Connection :
        Remote Party: Peter
        Remote Party: Andrew
        Supported Form Type : PatientRecord
                Sub Medium Type : summary
                Sub Medium Type : echo
        Supported Medium Type : audio streaming
        Supported Medium Type : text file

*********************************
Evaluating the next step
Number of events in queue 7
Event ID is CommActivitySchema_control/test1_r_aalle004.cmlcontrolsyntax
Not relevant for current workflow state, but could belong to a future state.
Still in Activity call A
Condition of Action is: true
Script Command produced :
Current state of negotiation : WaitingSameCS
Schema contents
*********************************
Local Party : Yali textFile LiveAudio AudioFile
Connection :
        Remote Party: Peter
        Remote Party: Andrew
        Supported Form Type : PatientRecord
                Sub Medium Type : summary
                Sub Medium Type : echo
        Supported Medium Type : audio streaming
        Supported Medium Type : text file
*********************************
Is more events queued false
Event ID is CommActivitySchema_control/test1_r_clarkep.cmlcontrolsyntax
Not relevant for current workflow state, but could belong to a future state.
Still in Activity call A
Condition of Action is: true
The updated schema is sent to UCI
Script Command produced :
sendSchema ("connection1" , "yali1028" , "Clarkep,Aalle004" , "test1.cmlcontrolsyntax" , "")
Current state of negotiation : NegComplete
Schema contents
*********************************
```

Figure A16: Kermeta Simulation Trace in Realizing Rainy Day Scenario - 1

```
Local Party : Yali textFile LiveAudio AudioFile
Connection :
        Remote Party: Peter
        Remote Party: Andrew
        Supported Form Type : PatientRecord
                Sub Medium Type : summary
                Sub Medium Type : echo
        Supported Medium Type : audio streaming
        Supported Medium Type : text file

*********************************
addParticipants("connection1","Clarkep,Aalle004")

About to switch schema~~~~
Event [cml::cmlWorkflowEvent::ConnSuccessful:21504] added with event ID being ConnectionSuccessful_connection1
Is more events queued true
Event ID is After(24.0)
-------------------Trigger events detected with ID After(24.0)
[cml::cmlWorkflowEvent::After:18863]
================Recognized as a Temporal Event
Next node call B
Activity call A is inactive...
Progress: Activity call B
Activity call B is executing...
Condition of Action is: true
Script Command produced :
sendSchema ("connection1" , "yali1028" , "Clarkep,Aalle004" , "test2.cmlcontrolsyntax" , "")
Current state of negotiation : WaitingSameCS
Schema contents
*********************************
Local Party : Yali textFile LiveAudio AudioFile VideoFile
Connection :
        Remote Party: Peter
        Remote Party: Andrew
        Supported Form Type : PatientRecord
                Sub Medium Type : summary
                Sub Medium Type : echo
        Supported Medium Type : audio streaming
        Supported Medium Type : text file
        Supported Medium Type : video file

*********************************
Evaluating the next step
Number of events in queue 5
Event ID is CommActivitySchema_control/test6_r_aalle004.cmlcontrolsyntax
Not relevant for current workflow state, but could belong to a future state.
Still in Activity call B
Condition of Action is: true
Script Command produced :
sendSchema ("connection1" , "yali1028" , "Clarkep,Aalle004" , "test2.cmlcontrolsyntax" , "")
Current state of negotiation : WaitingAnyCS
Schema contents
*********************************
Local Party : Yali textFile LiveAudio AudioFile VideoFile
Connection :
        Remote Party: Peter
        Remote Party: Andrew
        Supported Form Type : PatientRecord
                Sub Medium Type : summary
                Sub Medium Type : echo
        Supported Medium Type : audio streaming
```

Figure A17: Kermeta Simulation Trace in Realizing Rainy Day Scenario - 2

```
state of negotiation : NegReady
Schema contents

*********************************
Local Party : Yali textFile LiveAudio AudioFile VideoFile
Connection :
        Remote Party: Peter
        Remote Party: Andrew
        Supported Form Type : PatientRecord
                Sub Medium Type : summary
                Sub Medium Type : echo
        Supported Medium Type : audio streaming

*********************************
Is more events queued false                                    LII
Event ID is CommActivitySchema_control/test4.cmlcontrolsyntax
Not relevant for current workflow state, but could belong to a future state.
Still in Activity call B
Condition of Action is: true
selfRemoved event added
Script Command produced :
removeSelf()
sendSchema ("connection1" , "yali1028" , "Clarkep" , "test_rmSelf.cmlcontrolsyntax" , "")
sendSchema ("connection1" , "yali1028" , "Aalle004" , "test_rmSelf.cmlcontrolsyntax" , "")
Current state of negotiation : final
Schema Null!
Event [cml::cmlWorkflowEvent::ConnFailed:24369] added with event ID being ConnectionFailed_connection1

-------------------Trigger events detected with ID ConnectionFailed_connection1
Activity call B is inactive...
Workflow Execution Finished...
```

Figure A18: Kermeta Simulation Trace in Realizing Rainy Day Scenario - 3

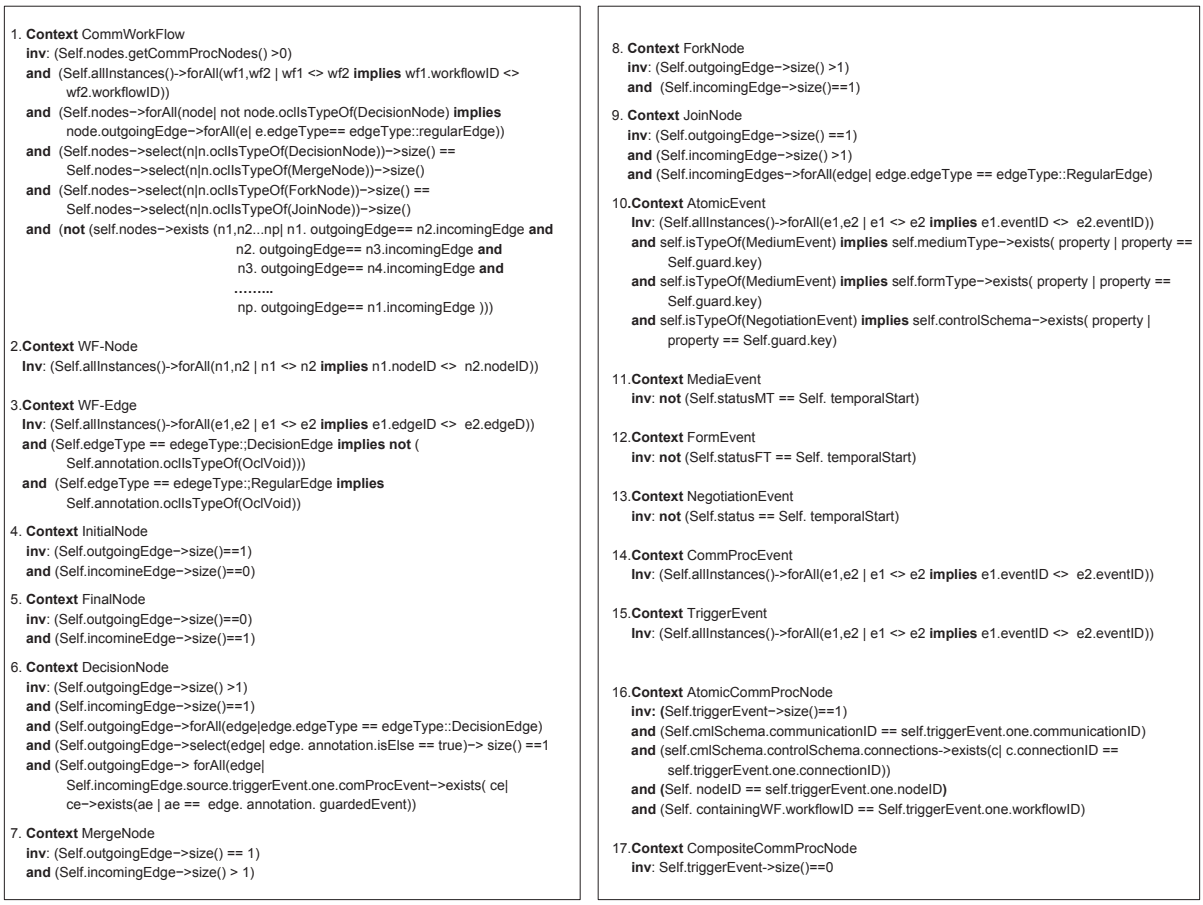

Figure A19: Static Semantics for WF-CML

Algorithm 6.1 Algorithm of Comm. Schema Controller for Synthesizing CML models.

```
 1: synthesisCML (ref (CI_in, DI_in))
 2: connSchemas ← decomposer.split((CI_in, DI_in))
 3: /*connSchemas contains a list of (CI_j,DI_j), each
    being an input instance pair for j^th connection */
 4: if (CI_in, DI_in) is from UCI then
 5:    new_conn ← getNewConn(Env_i.userSchema, connSchemas)
 6:    /* getNewConn returns the set of connections to be created */
 7:    if not (new_conn == null) then
 8:       for each conn ∈ new_conn do
 9:          conn_proc ← ConnProc.create(conn)
10:          addConnProc(Env_i, conn_proc)
11:          conn_proc.start()
12:       end for
13:    end if
14: end if
15: for each (CI_j, DI_j) ∈ ConnSchemas do
16:    conn_proc ← locateConnProc (j)
17:    exec_CML_CI(conn_proc, CI_j)
18:    exec_CML_DI(conn_proc, DI_j)
19: end for
20: old_conn ← getOldConn(Env_i.userSchema, connSchemas)
21: /* GetOldConn returns the set of connections to be torn down */
22: if not (old_conn == null) then
23:    for each conn in old_conn do
24:       conn_proc ← locateConnProc(j)
25:       exec_CML_CI(conn_proc,null)
26:       exec_CML_DI(conn_proc,null)
27:       /* clean up the state machines before tearing down the process */
28:       conn_proc.terminate()
29:    end for
30: end if

31: exec_CML_CI(ref conn_proc, ref CI)
32: CCI ← analyzer.analyzeCI(CI, conn_proc.Conn_Env_i, source)
33: /* Extract the difference in control instance and pass it
    to the Negotiation machine in the connection process */
34: conn_proc.Nego.stepNego(CCI)

35: exec_CML_DI(ref conn_proc, ref DI)
36: /* Assume a data schema carries one piece of media to be transferred */
37: CDI ← analyzer.analyzeDI(DI, conn_proc.Env_i,source)
38: conn_proc.MT.stepMT(CDI)
```

Algorithm 6.2 Algorithm to Analyze CML Control Schema.

```
1: analyze_CI (ref CI_{i+1}, ref Conn_Env_i, sourceCI)
   /*Input: CI_{i+1} - schema from the UCI or UCM
            Conn_Env_i - current environment object for a specific connection
            sourceCI - source of CI_{i+1}, UCI or UCM
     Output: cci, an object with CI changes and an event trigger */
2: cci ← compare(CI_{i+1}, Conn_Env_i.CI_i)
3: if sourceCI == UCI then
4:    if cci.enum ∈ {initialCI} then
5:       cci.addEvent(initiateNeg)
         /* SE Controller uses this event to start the state
         machines (SMs) for negotiation and media transfer
         and passes the initiateNeg to the negotiation SM */
6:    else if cci.enum ∈ {partyRemoved} then
7:       cci.addEvent(removeSelf)
8:    else if cci.enum ∈ {selfRemoved} then
9:       cci.addEvent(removeParty)
10:   else if not cci.enum ∈ {no_Change}} then
11:      cci.addEvent(initiateReNeg)
12:   end if
13: else if sourceCI == UCM && self.isInitiator then
14:   if cci.enum ∈ {partyRemoved} then
15:      cci.addEvent(removeParty)
16:   else if cci.enum ∈ {no_Change} then
17:      cci.addEvent(localSameCI)
18:   else
19:      cci.addEvent(localChangeCI)
20:   end if
21: else
22:   /*sourceCI == UCM && !self.isInitiator */
23:   if Conn_Env_i.CI_i == null then
24:      cci.addEvent(inviteNeg)
         /* CI Controller uses this event to start the state
         machines (SMs) for negotiation and media transfer
         and passes the inviteNeg to the negotiation SM */
25:   else if cci.enum ∈ {partyRemoved} then
26:      cci.addEvent(removeParty)
27:   else if cci.enum ∈ {no_Change} then
28:      cci.addEvent(remoteSameCI)
29:   else
30:      cci.addEvent(remoteChangeCI)
31:   end if
32: end if
33: return cci
```

Algorithm 6.3 Algorithm to Analyze CML Data Schema.

```
1: analyze_DI (ref DI_{i+1}, ref Conn_Env_i, sourceDI)
   /*Input: DI_{i+1} - schema from the UCI or UCM
            Conn_Env_i - current environment object
            sourceDI - source of DI_{i+1} is either UCI or UCM
     Output: - cdi, an object with DI changes and an event trigger */
2: cdi ← compare(DI_{i+1}, Conn_Env_i.DI_i)
3: if sourceDI == UCI then
4:    if cdi.enum ∈ {streamAdded} then
5:       cdi.addEvent(enableStream)
6:    else if cdi.enum ∈ {streamRemoved} then
7:       cdi.addEvent(disableStream)
8:    else if cdi.enum ∈ {nonStreamAdded} then
9:       cdi.addEvent(sendNonStream)
10:   else if cdi.enum ∈ {formAdded} then
11:      cdi.addEvent(sendForm)
12:   end if
13: else
14:   /*sourceDI == UCM*/
15:   if cdi.enum ∈ {streamAdded} then
16:      cdi.addEvent(enableStreamRec)
17:   else if cdi.enum ∈ {streamRemoved} then
18:      cdi.addEvent(disableStreamRec)
19:   else if cdi.enum ∈ {nonStreamAdded} then
20:      cdi.addEvent(recNonStream)
21:   else if cdi.enum ∈ {receiveForm} then
22:      cdi.addEvent(recForm)
23:   end if
24: end if
25: return cdi
```

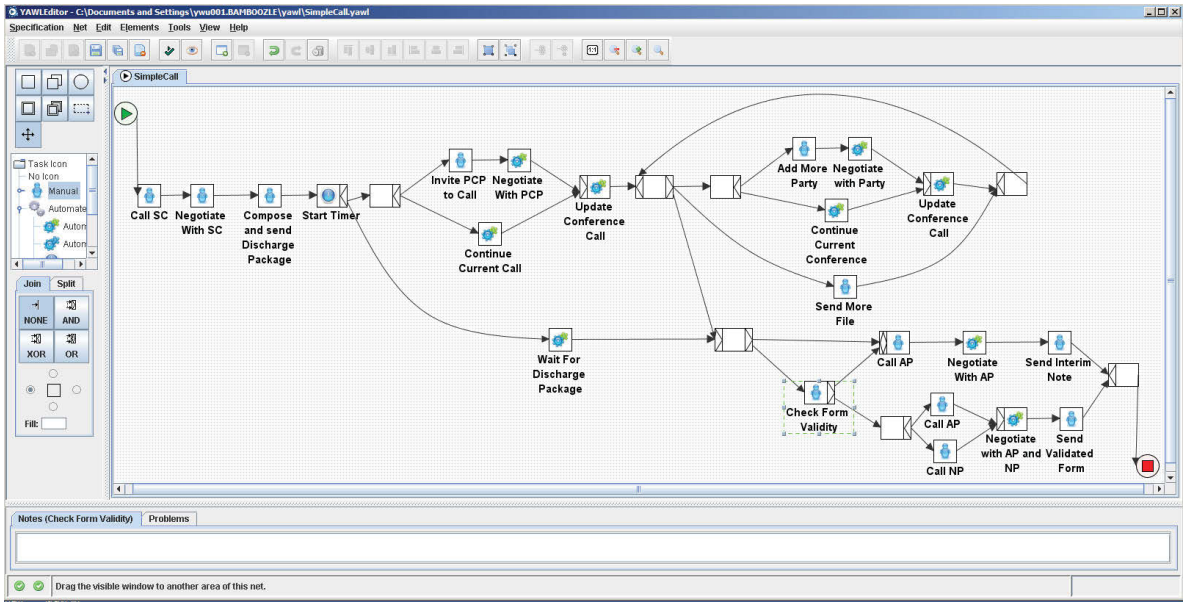

Figure A20: YAWL Process Editor for Creating YAWL Specification

Comparative Studies using YAWL and Microsoft WF

State Machines for Negotiation and Media Transfer

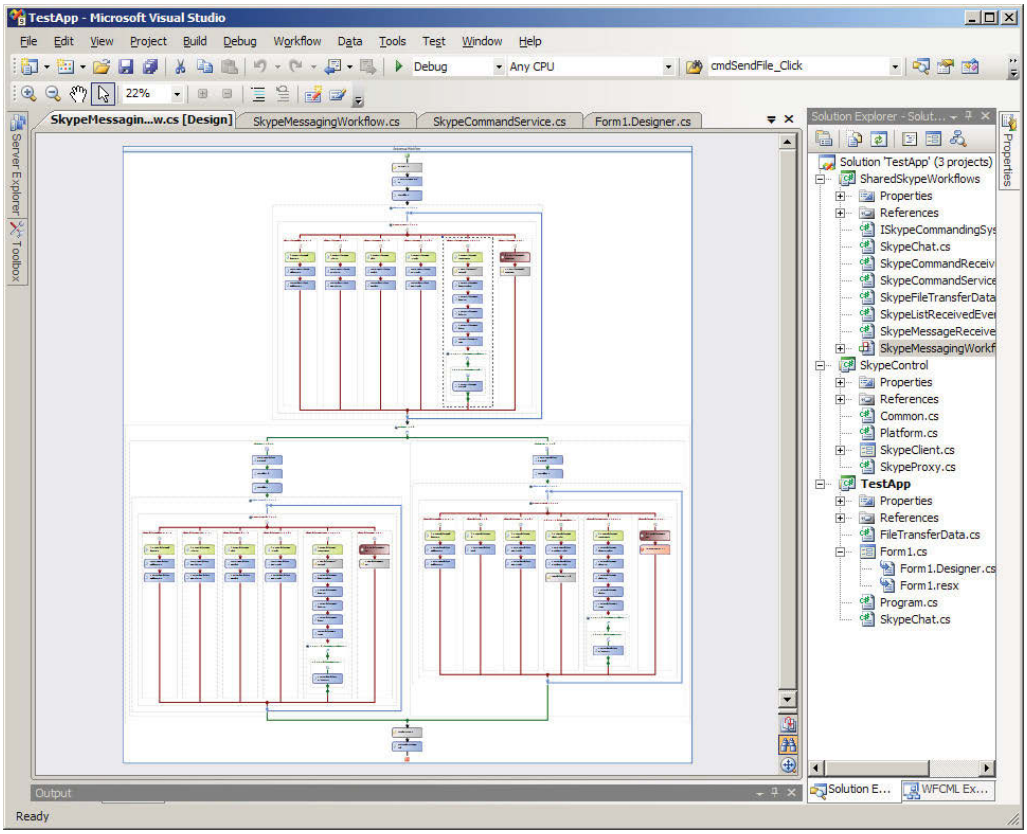

Figure A21: WF Process Editor for Creating WF Solution

Table A1: State machine for schema (re)negotiation.

T.	*Source_State*	*Target_State*	*Event*	*Guard*	*Action*
0	**Initial**	NegReady			
1	NegReady	NegInitiated	initiateNeg	hasNegToken	addNegBlock(CI_{neg}) genConnection_Script
2	NegReady	NegInitiated	initiateReNeg	hasNegToken	addNegBlock(CI_{neg})
3	NegInitiated	WaitingSameCI		# remoteParty != 0	genSendCS_Script
4	WaitingSameCI	WaitingSameCI	localSameCI	# responses < # remoteParty	
5	WaitingSameCI	NegComplete	localSameCI	# responses == # remoteParty	genSendCS_Script
6	NegComplete	NegReady			$CI_{exe} \leftarrow CI_{neg}$ UCI.notify(CI_{exe})
7	WaitingSameCI	WaitingAnyCI	localChangeCI	# responses < # remoteParty	update(CI_{neg})
8	WaitingSameCI	NegComplete	localChangeCI	# responses == # remoteParty	update(CI_{neg}) genSendCS_Script
9	WaitingAnyCI	WaitingAnyCI	localSameCI	# responses < # remoteParty	
10	WaitingAnyCI	WaitingAnyCI	localChangeCI	# responses < # remoteParty	update(CI_{neg})
11	WaitingAnyCI	NegComplete	localSameCI	# responses == # remoteParty	genSendCS_Script
12	WaitingAnyCI	NegComplete	localChangeCI	# responses == # remoteParty	update(CI_{neg}) genSendCS_Script
13	WaitingSameCI	WaitingAnyCI	after t sec.	# remoteParty > 1	update(CI_{neg})
14	WaitingAnyCI	WaitingAnyCI	after t sec.	# remoteParty > 1	update(CI_{neg})
15	WaitingSameCI	NegTerminateInit	after t sec.	# remoteParty == 1	
16	WaitingAnyCI	NegTerminateInit	after t sec.	# remoteParty == 1	
17	NegReady	NegRequested	inviteNeg		notifyUCI_InviteNeg
18	NegRequested	NegTermRemote	UCI.rejectInvite		genRejectInvite_Script
19	NegTermInit	**Final**			UCI.notify(CI_{exec}) genCloseConnect_Script
20	NegTermRemote	**Final**			
21	NegRequested	WaitingConfirm	UCI.acceptInvite		genConnection_Script genSendCS_Script
22	WaitingConfirm	NegComplete	remoteSameCI		
23	WaitingConfirm	WaitingConfirm	remoteChangeCI		update(CI_{neg}) genSendCS_Script
24	WaitingConfirm	NegTermRemote	after t sec.		UCI.notify(CI_{exe}) genCloseConnect_Script
25	NegReady	SelfRemoved	removeSelf	hasNegToken	genRemoveSelf_Script
26	NegReady	NegReady	removeParty	# remoteParty > 1	update(CI_{neg}) $CI_{exe} \leftarrow CI_{neg}$ genRemoveParty_Script
27	NegReady	**Final**	removeParty	# remoteParty == 1	genCloseConnect_Script UCI.notify(CI_{exe})
28	SelfRemoved	**Final**			

Table A2: State machine for media transfer.

Tr.	*Source_State*	*Target_State*	*Event*	*Guard*	*Action*
0	**Initial**	Ready	initiateNeg ‖ initiateInviteNeg		
1	Ready	StreamEnabled	enableStream		genStreamEnable_Script
2	Ready	StreamEnabled	enableStreamRec		genStreamEnableRec_Script UCI.notify(DS_{i+1})
3	StreamEnabled	StreamEnabled	enableStream	!IsStreamEnabled	genStreamEnable_Script
4	StreamEnabled	StreamEnabled	disableStream	IsStreamEnabled && # streams > 1	genStreamDisable_Script
5	StreamEnabled	StreamEnabled	enableStreamRec	!IsStreamEnabled	genStreamEnableRec_Script UCI.notify(DS_{out})
6	StreamEnabled	StreamEnabled	disableStreamRec	IsStreamEnabled && # streams > 1	genStreamDisableRec_Script UCI.notify(DS_{out})
7	StreamEnabled	StreamEnabled	sendNonStream		genNonStreamSend_Script
8	StreamEnabled	StreamEnabled	sendForm		genSendForm_Script
9	StreamEnabled	StreamEnabled	recNonStream		UCI.notify(DS_{out})
10	StreamEnabled	StreamEnabled	recForm		UCI.notify(DS_{out})
11	StreamEnabled	Ready	disableStream	# streams == 1	genCloseStream_Script
12	StreamEnabled	Ready	disableStreamRec	# streams == 1	genCloseStreamRec_Script UCI.notify(DS_{out})
13	Ready	Ready	sendNonStream		genNonStreamSend_Script
14	Ready	Ready	sendForm		genSendForm_Script
15	Ready	Ready	recNonStream		UCI.notify(DS_{out})
16	Ready	Ready	recForm		UCI.notify(DS_{out})
17	Ready	**Final**	terminate		

VITA

YALI WU

2006	B.S., Software Engineering Beihang University Beijing, China
2009	Computer Science Florida International University Miami, Florida
2011	Doctoral Candidate in Computer Science Florida International University Miami, Florida

PUBLICATIONS AND PRESENTATIONS

Yali Wu, Andrew A. Allen, Frank Hernandez, Robert France and Peter J. Clarke:*A Model-Driven Approach to Realizing User-Centric Communication Services*. SOFTWARE PRACTICE AND EXPERIENCE (journal accepted for publication 2011)

Yali Wu, Frank Hernandez, Robert France and Peter J. Clarke: *A DSML for Coordinating User-Centric Communication Services*. Proceedings of the 35th Annual IEEE International Computer Software and Applications Conference (COMPSAC 11)

Tariq M. King, Andrew A. Allen, Yali Wu, Peter J. Clarke, and Alain E. Ramirez: *A Comparative Case Study on the Engineering of Self-Testable Autonomic Software*. Accepted to the 8th IEEE International Conference on the Engineering of Autonomic and Autonomous Systems(Apr 2011)

Yali Wu, Frank Hernandez, Francisco Ortega, Peter J. Clarke and Robert B. France: *Measuring the Effort for Creating and Using Domain-Specific Models*. The 10th Workshop on Domain-Specific Modeling (DSM'10) in conjunction with SPLASH 2010. Oct. 17-18, 2010

Peter Clarke, Jairo Pava, Yali Wu, Tariq M. King. *Collaborative Web-Based Learning of Testing Tools in SE Courses*. Submitted to 42nd ACM Technical Symposium on

Computer Science Education, 2011. Accepted October 2010

Peter J. Clarke, Yali Wu, Andrew A. Allen and Tariq M. King: *Experiences of Teaching Model-Driven Engineering in a Software Design Course*. Proceedings of the Models 2009 Educators' Symposium (Oct 2009)

Yingbo Wang, Yali Wu, Andrew A. Allen, Barbara Espinoza, Peter J. Clarke and Yi Deng: *Towards the Operational Semantics of User-Centric Communication Models*. Proceedings of the 33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC 09)

Paola Boettner, Mansi Gupta, Yali Wu and Andrew A. Allen: *Towards Policy Driven Self-Configuration of User-Centric Communication*. Proceedings of the 47th ACM Southeast Conference (ACMSE 09)

Andrew A. Allen, Yali Wu, Peter J. Clarke, Tariq M. King and Yi Deng: *An Autonomic Framework for User-Centric Communication Services*. Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative research (CASCON 09)

Andrew A. Allen, Sean Leslie, Ricardo Tirado, Yali Wu and Peter J. Clarke: *Self-Configuring User-Centric Communication Services*. The Third International Conference on Systems (ICONS 2008)

Yali Wu, Andrew A. Allen, Frank Hernadez, Yingbo Wang and Peter J. Clarke: *A User-Centric Middleware for CVM*. Proceedings of the IASTED International Conference on Software Engineering and Applications (SEA 2008)

Yingbo Wang, Peter J. Clarke, Yali Wu, Andrew A. Allen and Yi Deng: *Runtime Models to Support User-Centric Communication*. Proceedings of the 3rd International Workshop on Models@runtime(Sept 2008)

Yingbo Wang, Peter J. Clarke, Yali Wu, Andrew A. Allen and Yi Deng: *Realizing Communication Services Using Model Driven Development*. Proceedings of the IASTED International Conference on Software Engineering and Applications (SEA 2007)