A DSML for Coordinating User-Centric Communication Services

Yali Wu, Frank Hernandez and Peter J. Clarke
School of Computing and Information Sciences
Florida International University
Miami, FL 33199, USA
{*ywu001, fhern006, clarkep*}*@cis.fiu.edu*

Robert France
Department of Computer Science
Colorado State University
Fort Collins, CO80532, USA
france@cs.colostate.edu

***Abstract*—**

Rapid advances in electronic communication devices and technologies have resulted in a shift in the way communication applications are being developed. The emerging development strategies provide end-users with a greater ability to manipulate the underlying communication technologies by providing the appropriate level of abstraction, referred to as *user-centric communication*. In communication-intensive domains such as telemedicine and disaster management, the user-centric communication strategies still lack the ability to coordinate the various communication services in collaborative processes.

In this paper, we present a domain-specific modeling language (DSML), *Workflow Communication Modeling Language* (WF-CML), that supports the rapid realization of collaborative user-centric communication applications. WF-CML is an extension of CML with communication specific abstractions of workflow concepts. To realize WF-CML models the dynamic synthesis process in the *Communication Virtual Machine* (CVM) prototype was extended to coordinate the negotiation and media transfer processes based on events generated during the collaboration. We also present a comparative study to show the advantage of using WF-CML over a general-purpose workflow language and execution environment.

***Keywords*-Model-Driven Development, Domain-Specific Modeling, Communication, Workflow Models,**

I. Introduction

Rapid advances in electronic communication devices and technologies, such as iPhone and Android, 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 that allow end-users to become more involved in the development of the so-called *user-centric communication applications*.

In communication-intensive domains such as telemedicine and disaster management, there is an increasing need for domain-specific communication applications that are user-centric and that support dynamic coordination of the various collaborating communication services[1], [2]. 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.

Existing approaches for realizing such applications usually involve the construction of customized communication solutions for specific domains. However, building such applications from scratch is a non-trivial task. They often rely on open communication APIs or frameworks for service enabling functionalities [3], [4], or a call modeling language such as CPL [5] for specifying coordinated telephony services. The popularity of Next Generation Networks has resulted in various user-centric service creation and delivery facilities [6], [7]. 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.

In this paper, we present a domain-specific modeling language (DSML), *Workflow Communication Modeling Language* (WF-CML), that supports the rapid realization of collaborative user-centric communication applications. We use the term user-centric 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. WF-CML is an extension of the previously developed *Communication Modeling Language* (CML) [8], [9] with additional constructs for dynamic coordination of user-centric communication services (UCCSs) in a collaborative environment. Models created using WF-CML are interpreted using an extended implementation of the Communication Virtual Machine (CVM) [10], 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. The contributions of this paper include:

- A metamodel defining the workflow communication modeling language (WF-CML).
- WF-CML semantics that enable the realization of coordinated UCCSs.
- A CVM prototype that supports the modeling and realization of coordinated UCCSs
- A comparative study between WF-CML and another workflow language to validate the claimed benefits.

We provide background on CML and CVM in Section II, and define the WF-CML in Section III. We describe how the CVM synthesizes WF-CML models to produce coordinated communication services in Section IV. Section V introduces the CVM prototype. Section VI presents a comparative study between WF-CML with another workflow language. Section VII describes related work and we conclude in Section VIII.

II. Preliminary Work and Motivation

In this section we give an overview of CML and CVM. A healthcare scenario is used to illustrate CML concepts and motivate the need for modeling the coordination of UCCSs in a collaborative environment.

A. Modeling and Realizing UCCSs

Communication Modeling Language (CML): CML was developed to raise the level of abstraction end-users model and realize communication services at runtime [11]. It is limited to modeling communication services between multiple participants and is designed to be simple and intuitive, yet expressive enough to model a majority of UCCSs.

There are currently two equivalent variants of CML: the XML-based (X-CML) and the graphical-based (G-CML). G-CML provides the expert user with a graphical way to define communication services for a specific domain, while X-CML is an internal presentation CVM manipulates. CML is used to describe *communication schemas* or *communication instances*, similar to the relationship between use cases and scenarios. To realize a communication service, two types of communication models are required: a *control schema (or instance)* that defines the configuration of the connections in a communication, and a *data schema (or instance)* that defines the media being transferred across a connection.

Communication Virtual Machine (CVM): CVM [10] provides a runtime environment that supports the modeling and realization of UCCSs specified in CML. Figure 1 shows the layered architecture of CVM. The CVM platform is divided into four major levels of abstraction, each layer playing a role in realizing communication services (see dashed horizontal lines in Figure 1). The layers of CVM are: (1) *User Communication Interface* (UCI) - provides a language environment for users to specify their communication requirements using CML; (2) *Synthesis Engine* (SE) - synthesizes CML models i.e., generates an executable script (*communication control script*) from a CML model and negotiates the model with other participants in the communication; (3) *User-centric Communication Middleware* (UCM) - executes the communication control script to manage and coordinate the delivery of communication services to users; (4) *Network Communication Broker* (NCB) - provides a network-independent API to UCM and works with the underlying network protocols or communication frameworks to deliver the communication services.

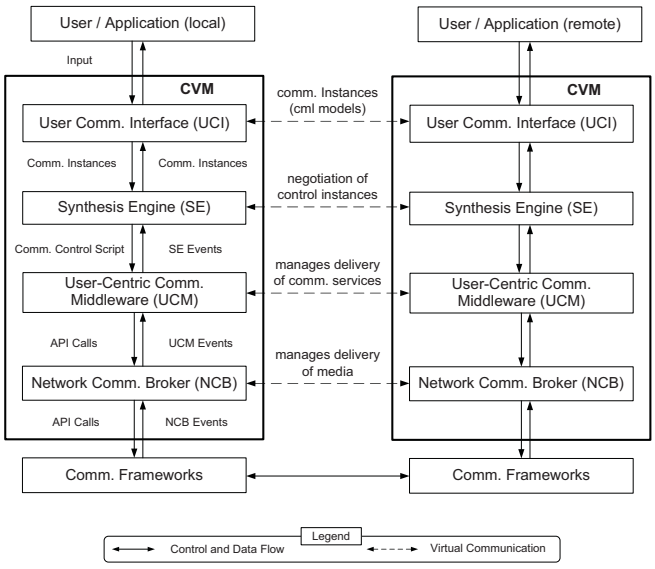

Figure 1. Layered architecture of the Communication Virtual Machine.

B. Motivating Scenario

We motivate the need for the class of user-centric communication applications that involved coordinated UCCSs using a real-world scenario provided by Dr. Burke, Director of Cardiovascular Surgery at Miami Children's Hospital (MCH). The scenario describes several communication sessions that take place when a patient is discharged. The actors in the scenario include: A discharge physician (DP), a senior clinician (SC), a primary care physician (PCP), a Nurse Practitioner (NP) and the Attending Physician (AP).

Motivating Example: 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 patient's condition (text file); x-Ray of the patient's heart (non-stream file); and an echocardiogram (echo) of the patient's heart (video clip). 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. If the 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 has to send out an interim discharge note (text file) to the AP. Meanwhile, Dr. Burke continues his conference with Drs. Monteiro and Sanchez. □

The scenario contains the communication between DP, SC and PCP, and the communication between DP and AP and NP. To achieve a more efficient collaborative process, the two communication instances need to be coordinated during the execution of the scenario. We show a G-CML control instance (CI) and data instances (DIs) in Figure

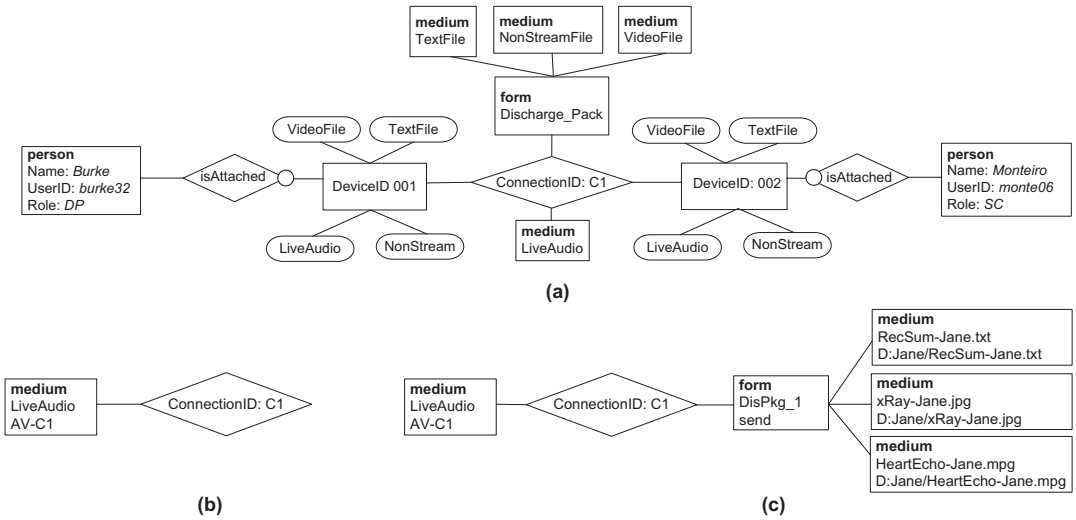

Figure 2. G-CML representation for: (a) the control instance for the 2-way call between Dr. Burke and Dr. Monteiro, (b) data instance to enable `LiveAudio`, (c) data instance to send the form `DisPkg_1`.

2(a) and 2(b) for the first communication instance. The existing CVM technology captures the communication needs of this scenario in separate CML instances[9], but automatic coordination of individual communication instances, often necessary in a collaborative process, is not supported. The lack of support for dynamic coordination of communication services motivates us to introduce communication-specific abstractions of workflow concepts into CML.

III. A DSML for Coordinated UCCSs

In this section we introduce WF-CML, a DSML for specifying the dynamic coordination of communication services. We briefly describe the language criteria and the meta-model of the DSML, and use the DSML to model the requirements for the healthcare scenario presented in Section II-B.

A. Language Criteria

The following criteria guided the development of WF-CML: The language should (1) be simple, yet expressive enough, to model the coordination of UCCSs by domain experts, (2) supports *realization* - dynamic synthesis and automatic execution of models, and (3) supports dynamic adaptation of executing models at runtime.

We reviewed several existing languages for modeling process coordination including UML activity diagrams [12], BPMN [13], BPEL [14] and YAWL [15]. Although these general-purpose languages could conceivably be used in place of WF-CML, the effort required to use them could be significantly greater than a DSML that provides the needed abstractions as first-class elements of the language [16]. Also, these languages produce complete workflow solutions, such as assigning activity ownerships and specifying the interaction between workflow engines and external applications, that are not always necessary [17]. For instance, we could integrate an existing workflow language with the CML. This allows us to use the workflow execution and analysis tools that come with the language. However, such a choice would certainly be desirable if we are looking for heavyweight support for workflows in CVM, which is currently not the case.

Note that since WF-CML targets at user-centric communication, it leaves out non-communication functionalities such as data acquisition and processing, data storage. ”Communication” as used in this paper denotes the exchange of electronic media of any format (e.g., file, video, voice) between a set of participants over a network (typically IP). This also results in a lightweight and agile workflow support in CVM through extending CML with necessary constructs rather than integrate a full-blown workflow modeling language. Furthermore, 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.

B. Meta-Model for WF-CML

We defined the metamodel of WF-CML using an abstract syntax, shown in Figure 3, and partial static semantics found on the project's website[1]. 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.

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

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

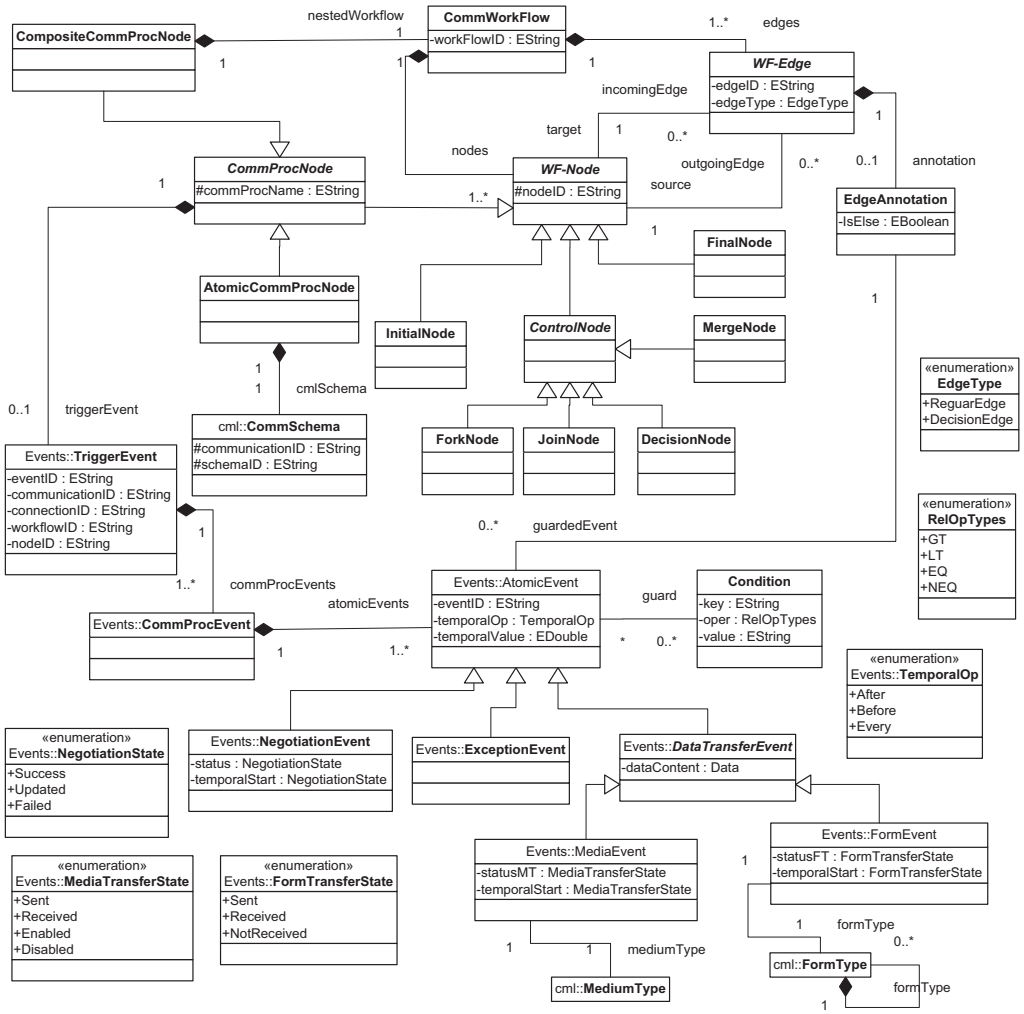

Figure 3. Abstract syntax for the WF-CML represented as a class diagram.

model has a CML model (`cml::Comm Schema`) and represents a communication service between participants, an example of which is shown in Figure 2. The meta-model for CML is also available on the project's website. 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 (`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`).

C. Modeling the Healthcare Use Case

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 healthcare scenario in Section II-B in Figure 4. The model includes three communication process nodes, each one containing a 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. A form type `Discharge_Pack` and a built-in media type `LiveAudio`

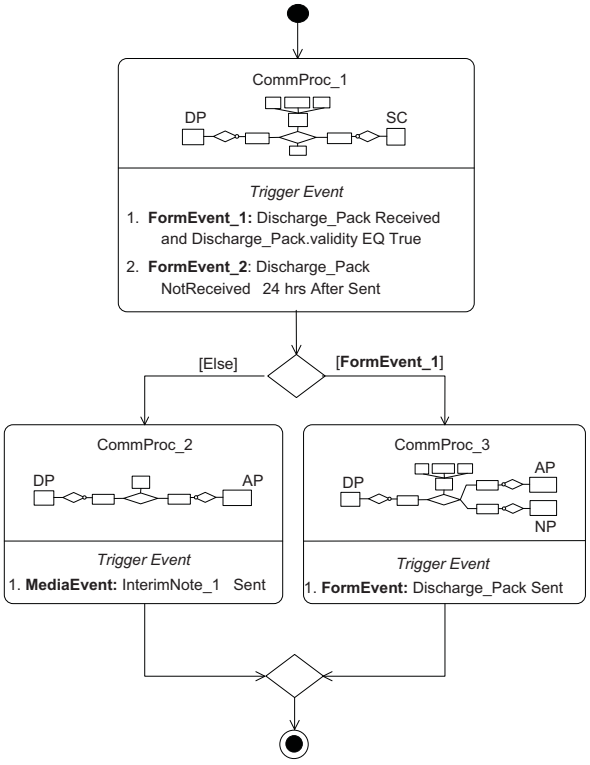

Figure 4. WF-CML model for healthcare use case.

is allowed for this communication. Note that the media type and form type are instantiated when Dr. Burke enables the audio stream, Figure 2(b), and loads the patient form `DisPkg_1`, Figure 2(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 (`DisPkg_1`); or the patient form is not received 24 hours after being sent.

IV. Realization of WF-CML Models

In this section we describe the semantic rules to support realization of CML and WF-CML models, and dynamic updates to the models at runtime. We use the term *realization* to refer to dynamic synthesis of the model into an executable communication control script, and automatic execution of the model through deploying the control script at runtime.

A. Overview of Realization

The semantic rules of WF-CML extend the semantic rules for CML [8]. We first provide an overview of the high-level semantic rules for realizing CML models followed by the high-level semantics rules for WF-CML models.

Realization of CML Models: The semantics to realize a CML model representing a communication service is captured as mappings from input CML models to output scripts and events, as well as changes in the environment. It is shown in the following transition rule.

$((\text{CI}_{in}, \text{DI}_{in}), \text{CSP\_Env}_i) \Longrightarrow ((\text{CI}_{out}, \text{DI}_{out}), \text{Script}_{out}, \text{Event}_{out}, \text{CSP\_Env}_{i+1})$

where:

- $(\text{CI}_{in}, \text{DI}_{in})$ - input control and data instances capturing user's communication needs to be realized by the communication service.
- CSP_Env_i - state of the CS process including the state of the executing control and data instances, $(\text{CI}_i, \text{DI}_i)$, negotiation state, Neg_i, and media transfer state, MT_i.
- $(\text{CI}_{out}, \text{DI}_{out})$ - updated control and data instances generated during the transition.
- Script_{out} - communication control script generated, including scripts for both (re)negotiation and media transfer.
- Event_{out} - output event generated during the execution of the CS process, including media events or negotiation events.
- CSP_Env_{i+1} - updated environment of the CS process. The structure is similar to CSP_Env_i stated above.

The behavior associated with a CS process is specified as a sequence of instance pairs of the form $(\text{CI}_i, \text{DI}_i)$, where $i = 0, 1 \cdots n$. The initial instance pair $(\text{CI}_0, \text{DI}_0)$ represents the initial state of the system with respect to some new connection to be established. CI_i specifies the intent of the end-user's communication needs using a declarative approach, whereas DI_i represents the actual data to be exchanged. Details of the labeled transition systems for (re)negotiation and media transfer, and the abstract syntax of the control script are explained by Wang [8].

Realization of WF-CML Models: The control flow for coordinating communication services is specified using an event-driven approach. The following execution rules are defined for each executing WF-CML model.

$(\text{Event}_{in}, \text{WF\_Env}_i) \Longrightarrow ((\text{CI}_{out}, \text{DI}_{out}), \text{WF\_Env}_{i+1})$

where:

- 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_Env_i - the current configuration of a process executing the WF-CML model (*WF_Proc*). Its state is defined as $(\text{WF}_{exec}, \text{CS\_Procs}, \text{Curr\_CS})$, where:
  - WF_{exec} - the currently executing WF-CML model in the WF_Proc process.
  - CS_Procs - a list of executing CS processes in the executing WF_Proc process.
  - Curr_CS - currently active CS processes with respect to the WF_Proc process.
- WF_Env_{i+1} - the updated configuration of the WF_Proc process.

B. Algorithms for Realizing WF-CML Models

To support the semantics rules described in the previous section we now describe the main data structure and algorithms used in the realization of WF-CML models.

WF-CML Hypergraph: We decided to use the hypergraph [18] since it provides for the removal of nodes (Decision, Merge, Fork and Join) that do not map to any real processes but are used mainly for modeling the routing rules. These control nodes can be replaced with "transitions", annotated edges, directly connecting communication process nodes. We defined WF-CML hypergraphs as follows:

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

where:

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

$\texttt{CS\_ProcNodes}$ - a set of atomic communication process nodes.

$\texttt{WaitNodes}$ - a set of wait nodes for handling synchronization for a join.

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

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

$\texttt{EdgeAnn}$ - events required to enable a transition.

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 [18]. 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 to create hyperedges, including concatenation of edge annotations.

Realization: The top level algorithm realize_WF, shown in Figure 5(a), 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 algorithm for analyze_WF is shown in Figure 5(b). 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 schema [8], 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*.

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. [18] 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 process to continue execution after a hypernode is exited. We omit the details of the hypergrpah traversal and focus on the differences previously stated.

Event mechanism: In Section III-B, 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. We assume that the events 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 are 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. [18] 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*. We expect to relax this strict definition of equivalence of CIs in the future.

V. CVM Prototype

We extended the CVM platform to handle WF-CML models. The CVM still maintains the layered architecture

```
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 scripts are generated dur-
      ing 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
```

(a)

```
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
```

(b)

Figure 5. Algorithms to (a) realize WF-CML, and (b) analyze WF-CML.

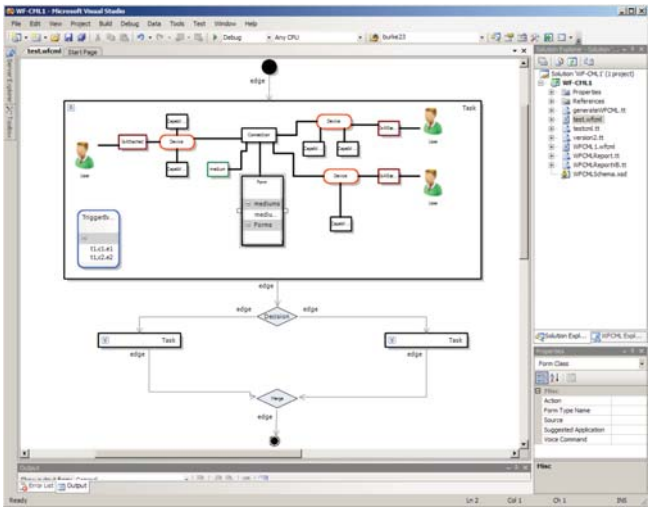

Figure 6. Modeling environment for coordinated UCCSs.

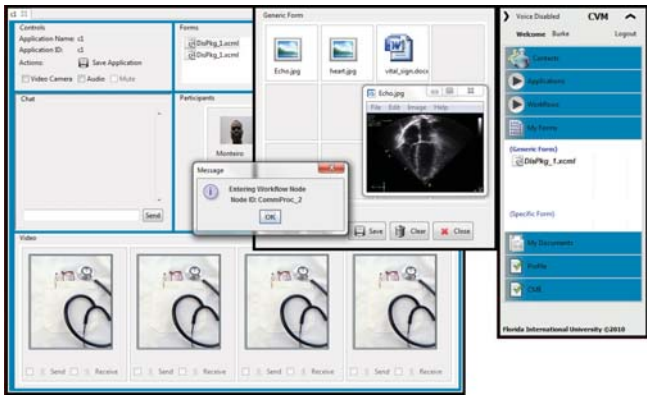

Figure 7. Dr. Burke's GUI showing (1) the message window requesting confirmation to advance in the workflow, and (2) Baby Jane's discharge package received from Dr. Monteiro.

as defined in Section II-A. 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 the Visual Studio DSL Tools [19]. Figure 6 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 III-C. Figure 7 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 8 shows the high-level design of the SE, including the flow of control when models are processed. The synthesis algorithm shown in Figure 5(a) is implemented in the *WF-CML controller*, shown on the left of Figure 8; similarly the analysis algorithm, Figure 5(b) is implemented in the *WF-CML analyzer*. Each executing WF-CML model has its thread of execution represented as $WFProc_K$ shown below the *WF-CML controller*. Events are received from (1) the SE dispatcher after the CI and DI are analyzed and the appropriate transitions made in the state machines for negotiation and media transfer, (2) internal timers initiated based on temporal properties related to the trigger events (not shown in the figure), and (3) from the UCM (external events). The events are managed by the *WF-CML controller* and dispatched to the $WFProc_K$s during execution. Each $WFProc_K$ is responsible for traversing and maintaining its own WF-CML Hypergraph.

VI. Comparative Study

In this section, we present a comparative study between WF-CML and YAWL (Yet Another Workflow Language)[15] to show the advantages of using a DSML versus a general purpose modeling language. YAWL is a

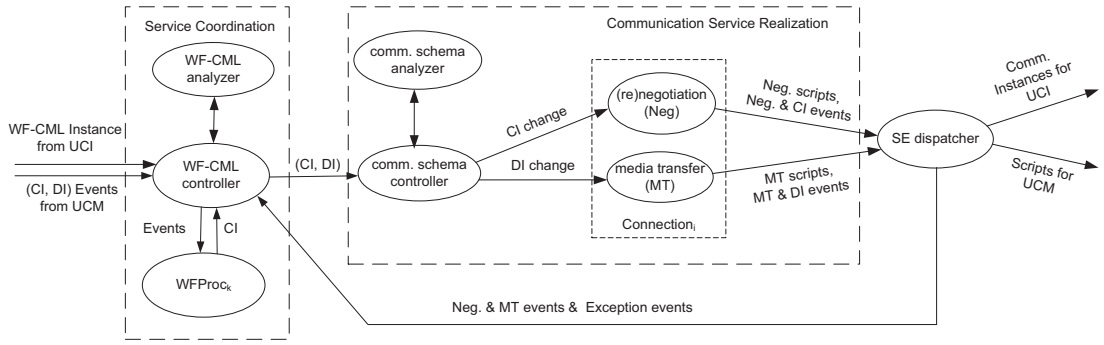

Figure 8. Execution of WF-CML communication instance in the synthesis engine (SE). CI - Control instance; DI - Data instance.

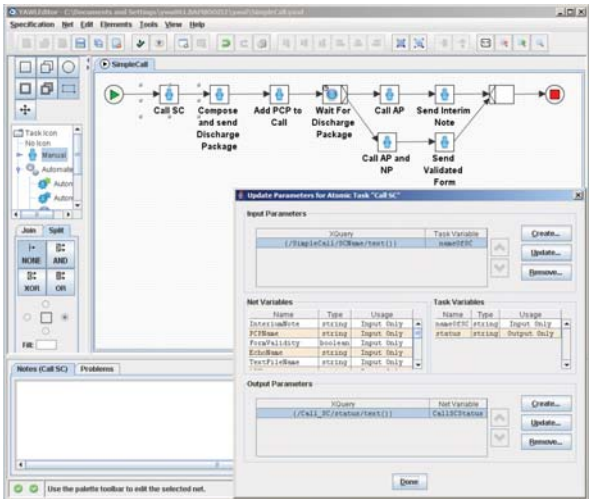

Figure 9. YAWL Process Editor for Creating YAWL Specification

concise and powerful workflow modeling language based on petri-nets and workflow patterns. It is supported by a business process management/workflow system that includes an execution engine and a graphical editor.

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. We chose YAWL in our study given its mature theoretical background, tooling support and easy access to its artifacts. We have started to explore a more comprehensive set of metrics to compare models developed using other DSMLs [17] e.g., BPEL [14] and Microsoft Workflow Foundation[20].

A. Environment Setup and Results

We modeled the same patient discharge scenario using both WF-CML (Figure 6) and YAWL(Figure 9) and 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, 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. The experimental steps are as follows:

- Set up both CVM and YAWL engine on Skype as the communication service provider for service realization
- Specify the same collaborative process using both WF-CML and YAWL in their respective graphical editors
- Load and execute service specifications in both CVM and YAWL engine

Table I shows the comparison of YAWL 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 II shows the collected static metrics of the YAWL implementation and CVM implementation, including number of lines of code, number of classes and methods. Dynamic metrics for the two implementations are also shown in Table II, 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.

B. Discussion

The results of the comparative study show that WF-CML has several advantages over YAWL in terms of the development effort, the ease of dealing with changing application needs and potential execution performance metrics.

Development Effort: Table I showed that in using WF-CML, modelers are shielded from service realization details of task nodes. These include the development and deployment of YAWL services, the definition of task and net variables, using parameter mapping to define data flow, and the definition of predicate guards for conditional branching. Using WF-CML the modeler is only required to specify the communication services using CML model and define trigger events. Therefore, WF-CML is more user-friendly than YAWL for domain experts (e.g., healthcare information specialist) who have limited expertise in programing.

Table I
COMPARISON BETWEEN WF-CML AND YAWL

Aspects of Comparison	*YAWL*	*CVM*
Setup Prerequisites	1. Servlet Container (Apache Tomcat default) 2. Database back-end (PostgreSQL default) 3. Skype4Java (Skype API for Java)	Skype4Java (Skype API for Java)
Process flow Specification	Create process flow in YAWL editor	Create process flow in WF-CML editor
Task/Service Specification	1. Develop and deploy web services that invoke Skype API calls 2. Register deployed web service with YAWL engine 3. Bind communication tasks in the YAWL specification to registered web services 4. Define task variables and net variables and mapping between input/output parameters to these variables	1. Specify communication services nodes using CML 2. 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 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	1. For changes in WF-CML process, update model in WF-CML editor 2. For changes in comm. service nodes, update CML model dynamically

Table II
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

Ease-of-Change: WF-CML is capable of responding to changing user communication needs, as well as changes in low level technology and implementation. Changes in the application needs are addressed at two levels: for basic communication services, users could dynamically update the communication without being restricted by the flow of control specified, as mentioned in Section VI-A. For changes in the process specification, developers only need to update the process model at design time without additional effort. To make such a change in the YAWL model, the modeler has to create the additional task, additional net variables and define the corresponding predicate guards.

Generality/Efficiency Trade-off: Table II provides evidence of 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 and models various workflow perspective, it is more heavyweight. Also, YAWL supports the interaction between running workflow instances with external applications exposed as services, which further contributes to its heavyweight nature. We trade generality for a lightweight workflow approach, demonstrated by the decrease in static complexity as well runtime overhead of the system.

VII. RELATED WORK

Modeling Communication: The idea of using domain-specific languages to specify robust and adaptable communication services has been around for sometime [5], [21]). The Call Processing Language (CPL)[5] is a DSL for programming Internet telephony services. It is used to quickly and safely specify call processing services such as call forwarding and anonymous call rejection. WF-CML differs from these DSLs in that DSLs like CPL are driven by abstractions over existing protocols or programming level solutions, and hence they are used by engineers to develop variations of a family of telephony services, whereas WF-CML uses the vocabulary of domain experts without being limited by technological considerations.

The emergence of Next Generation Networks has enabled user-centric service creation and delivery facilities, including OPUCE (Open Platform for User-centric service Creation and Execution) [6] and SPICE(Service Platform for Innovative Communication Environment [7]. These environments support the combination of Internet IT services with communication services such as presence and audio/video conferencing. However, 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.

Web Services and Service Coordination in SOA: WF-CML and the CVM framework share similarities with web services and SOA. Both approaches provide integration and coordination techniques to compose complex services from basic ones. By focusing on user-centric communication, using a DSL like WF-CML could achieve lightweight and agile communication process support, as opposed to heavy workflow support in regular workflow languages in SOA, such as YAWL[15], BPMN [13] and BPEL [14]. While these languages are designed to cover a broad spectrum of business processes, they are usually heavyweight (see comparative study). Also, these low-level languages are intended for use by software developers, whereas WF-CML targets domain experts and hence it is more user-friendly while enabling automatic realization.

Execution Semantics for Workflow Models: There has been

a plethora of work on defining the semantics for executable workflow models. Our work is most closely related to the work by Eshuis et al. [18], as described in Section IV-B. Eshuis et al. define the execution semantics for UML activity diagrams [12] using activity hypergraphs and label transition systems. We use some of the concepts describe by Eshuis et al. to convert WF-CML models to WF-CML hypergraphs, but adapted them accordingly to suit our semantic needs.

CML and CVM: In this paper we extended the work on CML presented by Clarke et al. [11] and Deng et al. [10] to include workflow constructs thereby allowing end-users to model the logical dependencies or coordinations between individual communication services, such as conference call or sending files. We also extended the semantics defined by Wang et al. [8] to handle dynamic synthesis of WF-CML models that allows the coordination of UCCSs.

VIII. Conclusion

In this paper, we presented a Workflow Communication Modeling Language (WF-CML), a DSML for modeling the coordination of user-centric communication services in a collaborative environment. A definition of the meta-model for WF-CML is given and the semantics for dynamically realizing WF-CML models are described. The Communication Virtual Machine (CVM) prototype was developed to support the creation and execution of WF-CML models. We also presented a comparative study between WF-CML and YAWL. Future work involves the development of policies e.g., security and optimization, that can be incorporated into WF-CML, and a more comprehensive comparative study.

Acknowledgment

This work was supported in part by the National Science Foundation under grants OISE-0730065, HRD-0833093, CCF-1018711; a Florida International University Dissertation Year Fellowship - Yali Wu; and a US Dept. of Ed. GAANN Fellowship (grant P200A070543) - Frank Hernandez.

References

[1] T. Y. Emily Carrier and R. A. Holtzwart, "Coordination between emergency and primary care physicians," 2011, national Institute For Health Care Reform. [Online]. Available: http://www.nihcr.org/ED-Coordination.html#section2

[2] L. K. Comfort, K. Ko, and A. Zagorecki, "Coordination in rapidly evolving disaster response systems," *American Behavioral Scientist*, vol. 48, no. 3, pp. 295–313, 2004. [Online]. Available: http://abs.sagepub.com/content/48/3/295.abstract

[3] T. P. Group, "Parlay/osa specifications," August 2010, http://etsi.org/WebSite/Technologies/OSA.aspx.

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

[5] C. U. Network Working Group, "Call Processing Language (CPL): A Language for User Control of Internet Telephony Services," October 2004, http://www.ietf.org/rfc/rfc3880.txt.

[6] J. C. Yelmo, J. M. del lamo, R. Trapero, and Y.-S. Martn, "A user-centric approach to service creation and delivery over next generation networks," *Computer Communications*, vol. 34, no. 2, pp. 209 – 222, 2011.

[7] S. Tarkoma, B. Bhushan, E. Kovacs, H. van Kranenburg, E. Postmann, R. Seidl, and A. 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*, June 2007, pp. 1–8.

[8] Y. Wang, Y. Wu, A. Allen, B. Espinoza, P. J. Clarke, and Y. Deng, "Towards the operational semantics of user-centric communication models," in *COMPSAC 09*. IEEE, July 2009, pp. 254–262.

[9] Y. Wu, A. A. Allen, F. Hernandez, R. B. France, and P. J. Clarke, "A domain-specific modeling approach to realizing user-centric communication," *Journal of Software Practice and Experience (SP&E)*, 2011, (to appear).

[10] Y. Deng, S. M. Sadjadi, P. J. Clarke, V. Hristidis, R. Rangaswami, and Y. Wang, "CVM - a communication virtual machine," *JSS*, vol. 81, no. 10, pp. 1640–1662, 2008.

[11] P. J. Clarke, V. Hristidis, Y. Wang, N. Prabakar, and Y. Deng, "A declarative approach for specifying user-centric communication," in *Proceeding of CTS 2006*. IEEE, May 2006, pp. 89 – 98.

[12] Object Management Group, "Unified modeling language: Superstructure, version 2," February 2009, http://www.omg.org/spec/UML/2.2/ (May. 2009).

[13] S. White, "Introduction to BPMN," IBM Corporation, Tech. Rep., 2004. [Online]. Available: http://www.bpmn.org/Documents/Introduction%20to%20BPMN.pdf

[14] IBM, "BPEL4WS, Business Process Execution Language for Web Services Version 1.1," 2003. [Online]. Available: http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf

[15] A. ter Hofstede, W. van der Aalst, and M. Dumas, "YAWL: Yet Another Workflow Language," April 2010, http://www.yawlfoundation.org/.

[16] N. Esfahani, S. Malek, a. P. Sousa, Jo H. Gomaa, and D. A. Menascé, "A modeling language for activity-oriented composition of service-oriented software systems," in *MODELS '09*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 591–605.

[17] Y. Wu, F. Hernandez, F. Ortega, P. J. Clarke, and R. France, "Measuring the effort for creating and using domain-specific models," in *Proceedings of 10th DSM Workshop*, 2010.

[18] R. Eshuis and R. 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*. London, UK: Springer-Verlag, 2001, pp. 47–61.

[19] S. Cook, G. Jones, S. Kent, and A. Wills, *Domain-specific development with visual studio dsl tools*. Addison-Wesley Professional, 2007.

[20] Microsoft Corporation, "Windows Workflow Foundation," July 2010, http://msdn.microsoft.com/en-us/netframework/aa663328.aspx.

[21] T. Margaria, B. Steffen, and M. Reitenspieß, "Service-oriented design: The roots," in *ICSOC*, 2005, pp. 450–464.