CEN 5011 Advanced Software Engineering

Section U01

Project Group 01

Professor: Peter J. Clarke

CVM GUI

FINAL DOCUMENT

Barbara Espinoza

Jorge Guerra

Eddie Incer

Ricardo Koller

David Martinez

Hong Soong

Nathanael Van Vorst

Date: 12/08/2008

Abstract

This document presents the results of the elaboration phase for developing the CVM GUI, the user interface of the Communication Virtual Machine (CVM). The CVM provides a model driven approach for developing communication applications and tools. It allows to declaratively specify communication services using a specific language called X-CML. The CVM also provides the functionalities for synthesizing and executing the communications specified in this language. The purpose of the CVM GUI is allowing end-users to specify and execute communication services by integrating with the Synthesis Engine layer of the CVM.

Based on the Unified Software Development Process we have implemented and tested our software system. This document merges together work products from all phases of the project. We present the use case model, describing the system's requirements and the analysis model that provides a semi-formal specification for these requirements. We also present the design model, which realizes the system requirements with a description of the software architecture and object design. The implementation phase is based on the blueprints provided by the design model. Finally, testing was performed at the unit, subsystem and system level. In this document we present the test suite developed for the CVM GUI together with the results of the test execution.

Table of Contents

TABLE OF FIGURES

1 INTRODUCTION

This chapter introduces the CVM GUI, and provides additional resources to aid in understanding the document. First section is the Purpose of The System, where we introduce our proposed system idea. Then the Scope of the System provides the system's domain. The Development Methodology follows the steps of Unified Software Development Process and applies it to our project. Then the Definitions and the Acronyms mostly used in our system's terminology are clarified. Finally, the Overview of the Document summarizes what this document entails.

1.1 PURPOSE OF THE SYSTEM

As technologies become pervasive in our lives, especially communication technology, there is an increasing need of accessing diverse forms of communication, such as texting, video conference, sound and file sharing from a single application. Thus, communication services are emerging to supply this rapid increase of demand; in particular, the Communication Virtual Machine (CVM) was conceived with that purpose in mind.

The CVM provides a model driven approach for developing communication applications and tools. It allows to declaratively specify communication services using a specific language called X-CML. The CVM also provides the functionalities for synthesizing and executing the communications specified in this language.

The CVM has been defined with a layered architecture composed of the following layers:

- The 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.
- The Synthesis engine (SE), which is a suite of algorithms to automatically synthesize a user communication schema instance to an executable form called communication control script.
- The user-centric communication middleware (UCM), which executes the communication control script to manage and coordinate the delivery of communication services to users, independent of the underlying network configuration.

- The 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 purpose of this project is building the CVM GUI, that is, the application that will allow end users to interact with the CVM. The CVM GUI includes both the UCI layer of the CVM and a GUI layer on top of that. The GUI layer allows users to communicate with other participants by, for example, sharing files, live audio/video streams or chat messages. The GUI then delegates the user requests to the UCI layer which in turn interacts with the Synthesis Engine for realizing the communication services.

Figure 1 depicts the layers in the CVM. At the top most there is the GUI, then the UCI a schema validation component and the repository. These four components form the CVM GUI. Then, below that are the lower layers of the CVM.

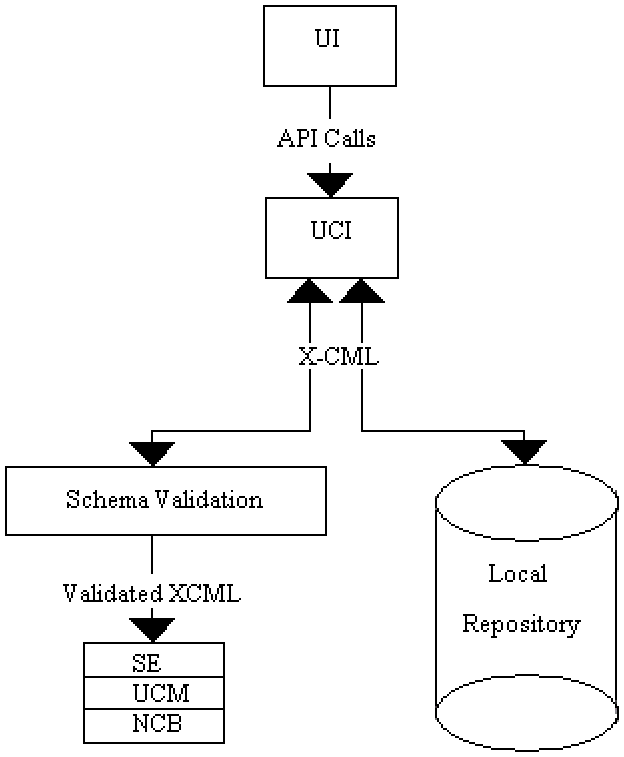


Figure 1 CVM Layers

1.2 Scope of the System

The CVM GUI provides the functionalities to load from a local repository and fully display the contents of schemas written in X-CML.

The CVM GUI should also provides the functionalities for saving communication schemas in X-CML format in the local repository.

Finally, the CVM GUI provides functionalities for dynamically updating the communication schema.

It is part of the scope of this project the implementation of the UCI as defined in the paper "A Communication Virtual Machine". This means that the UCI provides the user with the means to define and manage their communication schema, maintain the consistency between the views of participants and serve as the runtime for managing user sessions.

The scope of this project does not include the integration with an implementation of Synthesis Engine. Instead, the CVM GUI integrates with a stub of the Synthesis Engine in order to realize it functionalities. This means that the CVM GUI is not a fully functional user interface for the CVM with respect to data transfer.

1.3 Development Methodology

The Development Methodology used for the CVM GUI is based on the Unified Software Development Process (USDP). The activities in the USDP create and maintain a set of models of the software as illustrated by fig. 2 **Error! Reference source not found.**.

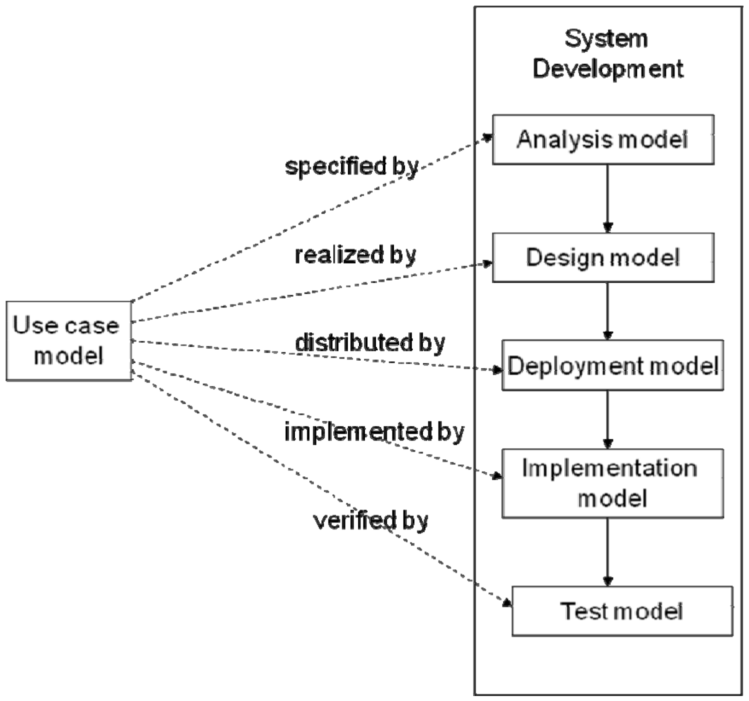


Figure 2 USDP Models

The Use Case model on the USDP captures the requirements and the Analysis model specifies those requirements using class and object interaction diagrams. At this point, a semi-formal specification of the system in terms of the application domain is available. Based on these two models, it is then possible to construct a Design model, which defines the structure of the system as a set of subsystems and interfaces. Additionally, as part of the design, the Deployment Model is constructed. The Deployment Model describes how the components of the system should be distributed.

The design of the CVM was built using the Software Requirements Document as an input. This document contains Use Case Model and a set of class and sequence diagrams that define the Analysis Model.

The Use Case Model, including the non-functional requirements, provides the input for defining the Software Architecture, which consists of the subsystem decomposition, the dependencies between subsystems, and the major policy decisions. Examples of policy decisions are mapping of software to hardware, global control flow and data storage requirements.

In a similar way, The Analysis model and the Use Case Model provide the input for the detailed design of the CVM GUI. This design consists of detailed class and sequence diagrams that extend the Analysis Model diagrams introducing solution domain classes and objects. Additionally, the CVM GUI detailed design includes state machine diagrams for the main control objects of each subsystem. These diagrams illustrate the behavior of the system in terms of the states that the control objects may be in and the transitions between those states.

The Deployment and Design Model then become realized by the Implementation Model. The software architecture and detailed design sets the foundation for the implementing phase. Both architectural and design patters are utilized to convey implemented classes interaction. The Use Case Model serves as a guide to what system use case functionality should be implemented by the Implementation Model. Subversion (SVN) was employed as a version control software to maintain current and historical source code files for the implementation phase.

Following the completion of the implementation, we proceeded to build the Test Model. The Test Model consists of test cases that show expected execution and outputs for given inputs. Testing was performed on the unit, subsystem and system levels. Unit testing was performed based on a State Machine specification for a major controller object in a subsystem. Subsystem testing verifies that two or more components interacted properly to produce a desired output. Finally System Test is conducted based on use case scenarios to test the main functionalities of the system. The Test Model is the final work product in our development process.

1.4 Definitions, Acronyms and Abbreviations

For more definitions, acronyms and/or abbreviations please see the glossary.

- CVM: Communication Virtual Machine.
- DD: Design Document.
- GUI: Graphical User Interface.
- SE: Synthesis Engine.
- SRD: Software Requirements Document.
- UCI: User Communication Interface.
- USDP: Unified Software Development Process.

1.5 Overview of the Document

The upcoming chapters of this document present some of the work products of the CVM GUI development project. The content of each section is as follows:

- Chapter 2 describes the current system.

- Chapter 3 presents the project plan, that is, the team organization, work breakdown for each of the project phases and, the hardware and software requirements of the project.
- Chapter 4 presents the system requirements for the CVM GUI together with the use case diagrams and a summary of the Analysis Model.
- Chapter 5 describes the software architecture of the CVM GUI, in particular the subsystem the composition, the hardware and software mapping and the persistent data management strategies.
- Chapter 6 explains the detailed design of the CVM GUI using UML Class, Sequence and Statechart diagrams.
- Chapter 7 presents the system, subsystem and unit test suites defined for the CVM GUI, as well as the test results.
- Chapter 8 is a glossary of the most important terms used in this document.
- Chapter 9 shows the approvals obtained for this document.
- Chapter 10 contains a list of appendixes showing the project schedule, use cases, user interface screenshots, detailed class diagrams, implementation of the test driver and meeting diary.

2 CURRENT SYSTEM

Currently there exists a functional prototype of the CVM GUI. This prototype provides functionalities for streaming live audio/video, sending chat messages, sharing files and, sharing forms. It allows users to manage their accounts, in particular, manage their contact list and change their profiles. It also allows users to create generic and specific forms. Generic forms consist of a set of files. Specific forms, on the other hand, are created by using a mediator application that is integrated into the prototype. The mediator application retrieves data from external data sources and generates a form based on that data.

The main issue with the current prototype is that it is not integrated with the lower layers of the CVM. This means that X-CML is not being used to declaratively define communication schemas. It also means that X-CML communication service specifications are not being exchanged between the CVM GUI and the Synthesis Engine. Finally, this implies that the prototype is not updating the communication schema based on negotiated schemas provided by the Synthesis Engine.

3 Project Plan

3.1 Project Organization

The project role assignment for the CVM GUI project is as follows:

Phase I - Inception

Team Member	Roles
Barbara Espinoza	Systems Analyst, Document Editor, Document Reviewer, Minute Taker
Jorge Guerra	Systems Analyst, Developer, Document Reviewer, Time Keeper
Eddie Incer	Systems Analyst, Document Editor, Team Lead
David Martinez	Systems Analyst, Developer
Ricardo Koller	Systems Analyst, Developer
Hong Ming Soong	Systems Analyst, Developer
Nathanael Van Vorst	Systems Analyst, Developer

Phase II - Elaboration

Team Member	Roles
Barbara Espinoza	Designer, Developer, Team Lead
Jorge Guerra	Designer, Developer
Eddie Incer	Designer, Developer
David Martinez	Designer, Developer, Document Editor, Minute Taker
Ricardo Koller	Designer, Developer, Time Keeper
Hong Ming Soong	Designer, Developer, Document Editor
Nathanael Van Vorst	Designer, Developer, Architect

Phase III - Construction

Team Member	Roles
Barbara Espinoza	Developer, Test Analyst, Document Editor
Jorge Guerra	Developer, Test Analyst, Minute Taker
Eddie Incer	Developer, Test Analyst, Time Keeper, Technical Writer
David Martinez	Developer, Test Analyst
Ricardo Koller	Developer, Test Analyst, Document Editor
Hong Ming Soong	Developer, Test Analyst, Test Manager
Nathanael Van Vorst	Developer, Test Analyst, Team Lead

3.2 Hardware and Software Requirements

3.2.1 Hardware Requirements

Every workstation must be equipped with at least:

- 1 GHz 32-bit (x86) processor or similar.
- 1 GB of system memory.
- 15 GB of available storage space.
- Internet Access.

3.2.2 Software Requirements

The following software is required on all workstations:

- Web Browser.
- PDF Reader.

For documentation and design activities the following software is required:

- Windows XP or Vista.
- Microsoft Office XP or newer.
- StarUML.
- Microsoft Project.
- COCOMO II.2000.0

For development and testing activities the software below is required:

- Java SDK 6.
- Eclipse 3.4.
- JUnit (Included with Eclipse).
- Eclipse SWT 3.4.
- Subclipse.

3.3 WORK BREAKDOWN

This section outlines the main activities, work products and milestones for each phase of the project: Inception, Elaboration and Construction. See **Error! Reference source not found.** for the project schedule and the Gantt chart of the project.

3.3.1 INCEPTION PHASE

Activities

Activity	Description
Project Planning	Involves determining the project organization, defining communication mechanisms, establishing the work breakdown, building the project schedule, identifying project risks and estimating project costs.
Project Monitoring	For the project monitoring weekly team meetings will be scheduled.
Requirements Elicitation	Includes understanding the application domain, understanding the current system, defining the purpose of the system, identifying the actors, defining the core use cases, capturing non-functional requirements, outlining the system scope and identifying the hardware and software requirements.
Analysis	The Analysis for this project involves creating Use Case Diagrams, identifying a set of main usage scenarios and building Object and Sequence diagrams based on them.
GUI Prototype Development	As part of the Inception phase a non-functional GUI Prototype is going to be implemented using the technologies selected for the system. This prototype is expected to evolve into the CVM GUI.
SRD Development	Based on the outputs from other activities in the inception

	phase the Software Requirements Document will be assembled.
Presentation Preparation	Presentation slides for the results of the inception phase will be prepared, divided among the team members and rehearsed.

Work Products

The following work products are identified for this phase:

- Project Plan.
- Meeting Minutes.
- Project Cost Estimate.
- Project Definition (Includes Project Purpose and Scope).
- Description of the Current System.
- Use Case Model.
- Analysis Model.
- Non-functional GUI Prototype.
- Project Glossary.
- Presentation Slides (Deliverable).
- SRD (Deliverable).

Milestones

- SRD Delivery and Presentation: marks the finalization of the Inception Phase.

3.3.2 Elaboration Phase

Activities

Activity	Description
Project Planning	The project schedule should be updated for the construction phase as the system Design Model is created.

Project Monitoring	For the project monitoring weekly team meetings will be scheduled.
Analysis	As part of the Elaboration Phase a summarized list of the functional and non functional requirements will be generated using the format "The System Shall…"
System Design	The system design will be based on the Use Case and Analysis Models. As part of the System Design, an Architectural Model including architectural patterns, subsystem decomposition, hardware and software mapping and storage requirements should be defined.
Detailed Design	For the detailed design of the CVM GUI, class, state machine and sequence diagrams should be generated. The detailed design for each class should be specified and design patterns must be used when suitable.
Implementation	For the purpose of the elaboration phase the non-functional prototype generated during the inception phase will be extended with the implementation of interfaces designed for each subsystem.
DD Development	Based on the outputs from other activities in the elaboration phase the Design Document will be assembled.

Work Products

The following work products are identified for this phase:

- Updated Project Plan.
- Meeting Minutes.
- Summarized list of Requirements.
- Updated Use Case Model.
- Design Model.
- Deployment Model.
- GUI Prototype extended with the implementation for the Subsystem Interfaces.
- Updated Project Glossary.

- DD (Deliverable).

Milestones

- DD Delivery: marks the finalization of the Elaboration Phase.

3.3.3 Construction Phase

Activities

Activity	Description
Project Monitoring	For the project monitoring, weekly team meetings will be scheduled.
Design	The design and deployment models must be updated based on any changes applied during the implementation of the CVM GUI.
Implementation	During the construction phase the requirements selected for the scope of this project will be implemented by extending the GUI Prototype produced during the Elaboration Phase.
Testing	A set of System and Subsystem Test Cases will be generated. Unit tests must be developed as well as testing stubs for allowing subsystems to be tested independently. A test driver must also be created for testing subsystems. The output for this activity will also include a report of the test results.
FD Development	Based on the output from all the activities in the construction phase and all previous phases, the Final Document will be assembled.
User’s Guide Development	A user’s guide for the CVM GUI should be created.
Presentation Preparation	Presentation slides for the results of the CVM GUI project will be prepared, divided among the team members and rehearsed.

Work Products

The following work products are identified for this phase:

- Meeting Minutes.
- Updated Design Model.

- Updated Deployment Model.
- CVM GUI implementation (Deliverable).
- Set of System and Subsystem Test Cases.
- Test Stubs and Drivers (Deliverable).
- Unit Tests (Deliverable).
- User’s Guide (Deliverable).
- Presentation Slides (Deliverable).
- FD (Deliverable).

Milestones

- Final Presentation and System Delivery: marks the finalization of the project.

4 REQUIREMENTS OF THE SYSTEM

This chapter presents the functional requirements of the system together with their associated non-functional requirements, a set of UML Use Case diagrams describing the Use Case Model of the CVM GUI, and a summary of the Analysis Model.

4.1 FUNCTIONAL AND NON-FUNCTIONAL REQUIREMENTS

The following is a list of the requirements of the system with their associated non-functional requirements and use cases.

1. The system shall provide the user with the ability to create, load, save, join, leave and update communications by interacting with a graphical user interface.
   Non-Functional Requirements:

   - A user without experience shall be able to create a new communication in less than 2 minutes.
   - A user without experience shall be able to load a communication in less than 2 minutes.
   - A user without experience shall be able to join a communication in less than 15 seconds.
   - The system shall finish creating a new communication in less than 100 milliseconds.
   - The system shall finish loading a communication schema in less than 1 second.
   - The Synthesis Engine should be informed of the user's decision no later 100 milliseconds after the user indicates it.
   - The system should display the negotiated communication in less than 5 seconds after it is received from the Synthesis Engine.
   - The system shall gracefully recover 100% of the times a communication schema is invalid.

   Related Use cases:

   - *CVMGUI_COM_001* - Create Communication
   - *CVMGUI_COM_002* - Load Communication
   - *CVMGUI_COM_005* - Save Communication

- *CVMGUI_COM_003* - Join Connection
- *CVMGUI_COM_004* - Leave Communication
- *CVMGUI_COM_009* - Update Communication.

2. The system shall provide the user with the ability to add participants from their contact list to active communications.

   Non-Functional Requirements:

   - A user without experience shall be able to add a participant within 30 seconds.
   - The system shall finish adding the participant in less than 3 seconds.

   Related Use cases:

   - *CVMGUI_COM_006* - Add Participant.

3. The system shall provide the user with the ability to create generic forms based on an existing set of files.

   Non-Functional Requirements:

   - A user without experience shall be able to create a generic form within 1 minute.
   - The system shall finish creating a new generic form in 10 milliseconds times the number of files added.

   Related Use Cases:

   - *CVMGUI_MED_007* - Create Generic Form

4. The system shall provide the user with the ability to enable and disable the Live Audio/Video Medium on an active communication.

   Non-Functional Requirements:

   - A user without experience shall be able to enable live audio/video in less than 1 second.
   - A user without experience shall be able to disable live audio/video in less than 1 second.
   - The system shall finish enabling live audio/video in less than 2 seconds.
   - The system shall finish disabling live audio/video in less than 2 seconds.

   Related Use Cases:

   - *CVMGUI_MED_002* - Enable Live Audio/Video Medium

- *CVMGUI_MED_003* - Disable Live Audio/Video Medium.

5. The system shall allow the user to share various types of media with the participants in the communication. In particular, it shall allow sharing files, live audio/video streams, chat messages, generic forms and specific forms.

   Non-Functional Requirements:

   - A user without experience shall be able to share a file in less than 1 second
   - The system shall finish sharing a file in less than 2 seconds.
   - A user without experience shall be able to start live audio/video in less than 1 second.
   - The system shall finish starting live audio/video in less than 2 seconds.
   - A user without experience shall be able to send a chat message in less than 1 second.
   - The system shall finish sending chat message in less than 2 seconds.
   - A user without experience shall be able to share a generic form within 30 seconds.
   - The system shall finish sharing a generic form in 100 milliseconds times the number of files in the form.
   - A user without experience shall be able to share a specific form within 30 seconds.
   - The system shall finish sharing a specific form in 100 milliseconds times the number of files in the form.

   Related Use Cases:

   - *CVMGUI_MED_001* - Share Media
   - *CVMGUI_MED_004* - Share File
   - *CVMGUI_MED_005* - Start Live Audio/Video
   - *CVMGUI_MED_006* - Send Chat Message
   - *CVMGUI_MED_008* - Share Generic Form
   - *CVMGUI_MED_009* - Share Specific Form

6. The system shall provide the users with the ability to manage their account. This includes starting the application, starting a session, finalizing a session, creating an account, updating an account, recovering their password, adding contacts, and and removing contacts.

   Non-Functional Requirements:

   - The user should be able to start the system in 1 second.
   - The system should display the GUI within 5 seconds of the start of the system.
   - The user should be able to login in 5 seconds.
   - The system should complete the login procedure within 5 seconds.
   - The user should be able to logout in 1 second.
   - The system should complete the logout procedure within 5 seconds.
   - The user should be able to create an account within 3 minutes.
   - The Create new account window should display within 1 second of pressing the Create new account button. The system should validate the user information within 3 seconds.
   - The user should be able to create an account within 3 minutes.
   - The Update user account window should display within 1 second of pressing the Update account button. The system should update the user information within 3 seconds.
   - The user should be able to recover his/her password in 10 seconds.
   - The system should display the recover password GUI within 1 seconds of clicking the Recover password button.
   - The recover password functionality should be available 95% of the time.
   - A user without experience shall be able to add contact in less than 1 second.
   - The system shall finish adding contact in less than 2 seconds.
   - A user without experience shall be able to remove a contact in less than 5 seconds.
   - The system shall finish removing the contact in less than 1 second.

   Related Use Cases:

   - *CVMGUI_ACC_001* - Start Application

- *CVMGUI_ACC_002* - Login
- *CVMGUI_ACC_003* - Logout
- *CVMGUI_ACC_004* - Create Account
- *CVMGUI_ACC_005* - Update Account
- *CVMGUI_ACC_006* - Recover Password
- *CVMGUI_ACC_007* - Add Contact
- *CVMGUI_ACC_008* - Remove Contact.

7. The system shall block users after three failed login attempts, prevent users from adding themselves as contacts, and prevent users from adding duplicate participants in a communication.

   Non-Functional Requirements:

   - It should be obvious to first time users when they have failed to login and why the account is locked.
   - The CVM should notify the user that login attempt failed within 2 seconds from receiving a response from the account management system.
   - The CVM should never login a user without verifying the username and password with the account management system.
   - A user without experience shall be able to add a new contact in less than 5 seconds.
   - The system shall respond the user that he cannot add himself as a contact in less than 2 seconds.
   - A user without experience shall be able to add a participant within 30 seconds.
   - The system shall gracefully recover 100% of the times when a participant is already in the communication.
   - The system shall finish checking the participants of the communication in 10 milliseconds.

   Related Use Cases:

- *CVMGUI_SEC_001* - Block User.
- *CVMGUI_SEC_002* - Prevent Self Contact.
- *CVMGUI_SEC_003* - Prevent Duplicate Participant

8. The system shall validate communications, remove invalid participant, verify single session and verify single instance.

   Non-Functional Requirements:

   - It should take less than 500 milliseconds to validate a communication schema.
   - System shall be able to verify single session in less than 1 second.
   - The system shall finish verifying single session in less than 2 seconds.
   - System shall be able to verify single instance in less than 1 second.
   - The system shall finish adding the user in less than 2 seconds.
   - System shall check invalid participant when loading communication in less than 1 second.
   - The system shall finish checking invalid participants when loading communication in less than 2 seconds.

   Related Use Cases:

   - *CVMGUI_ACC_004* - Validate Communication.
   - *CVMGUI_ACC_005* - Remove Invalid Participant.
   - *CVMGUI_ACC_006* - Verify Single Instance.
   - *CVMGUI_ACC_007* - Verify Single Session.

4.2 USE CASE DIAGRAMS

We divided the set of CVM GUI use cases into four packages: Communication, Media, Account and Security. This organization is depicted in Figure 3. The contents of these packages are as follows.

- The Communication package contains those use cases related with initiating or updating communication schemas. Figure 4 shows a use case diagram for this package.

- The Media package contains the functionalities related with sharing media within a communication. Figure 5 shows a use case diagram for this package.
- The Account package contains the use cases related with managing the user's account information. Figure 6 shows a use case diagram for this package.
- The Security Package contains a set of security use cases specified for the system. Figure 7 shows a use case diagram for this package.

It should be noted that the scope of the design included only those use cases in the Communication and Media packages.

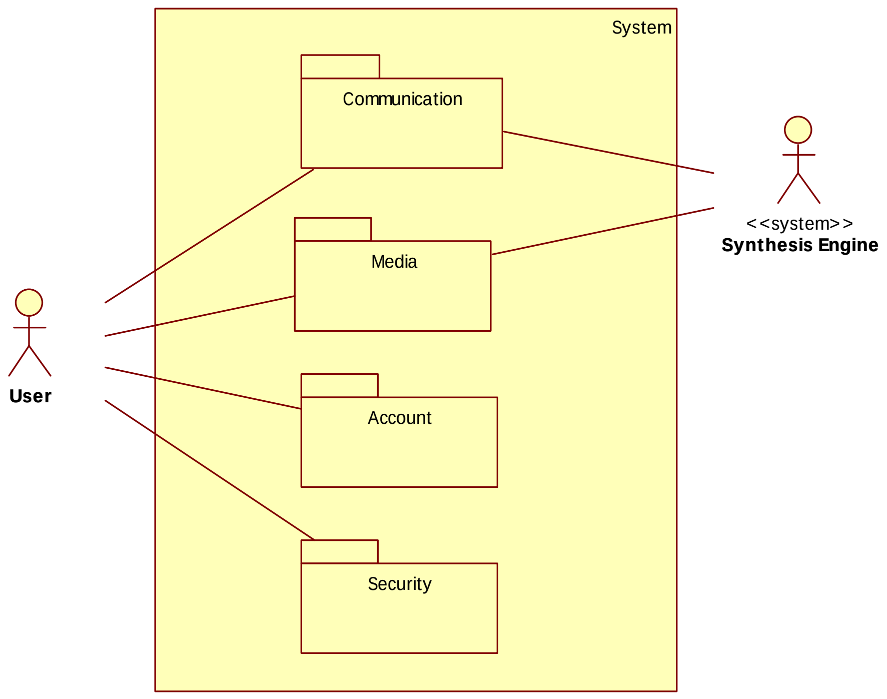


Figure 3 Use Case Packages & Actors

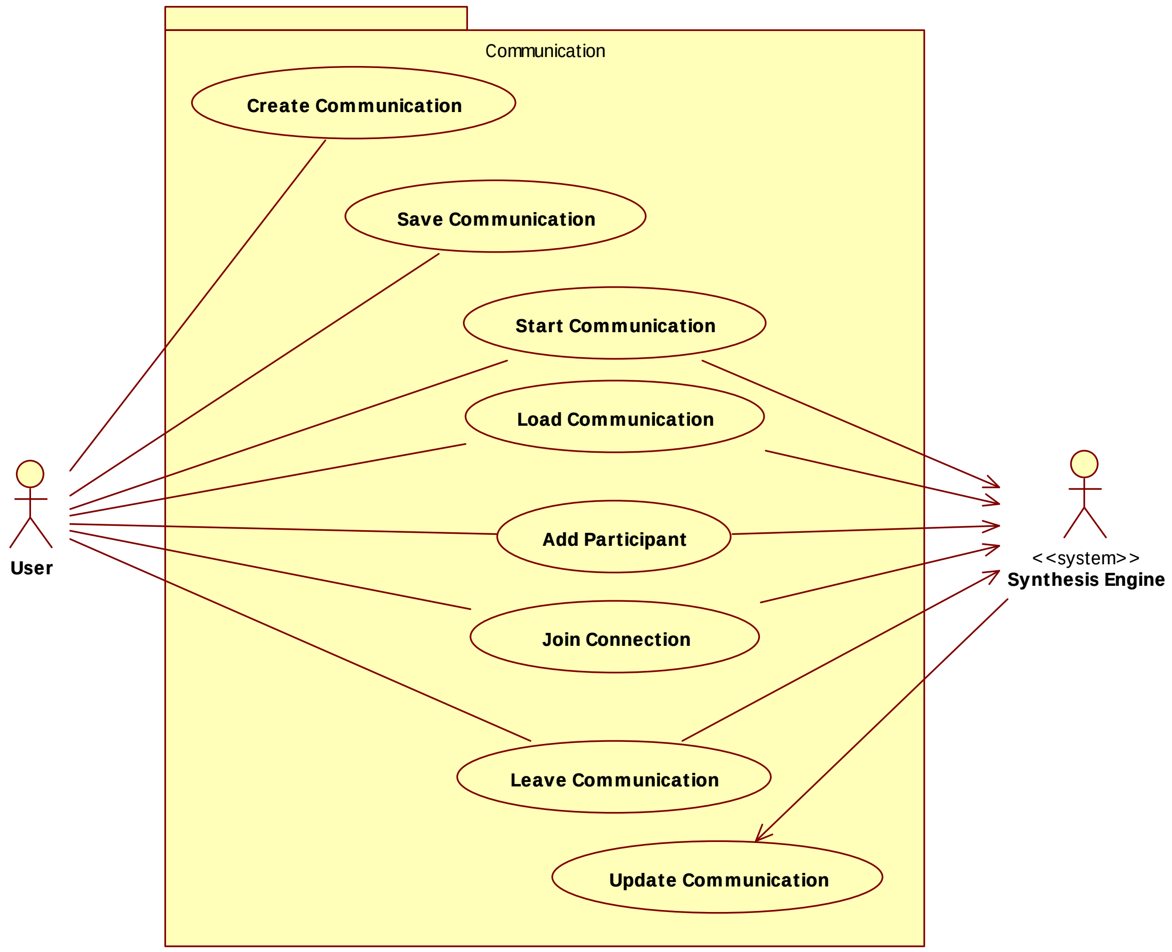


Figure 4 Communication Package Use Cases

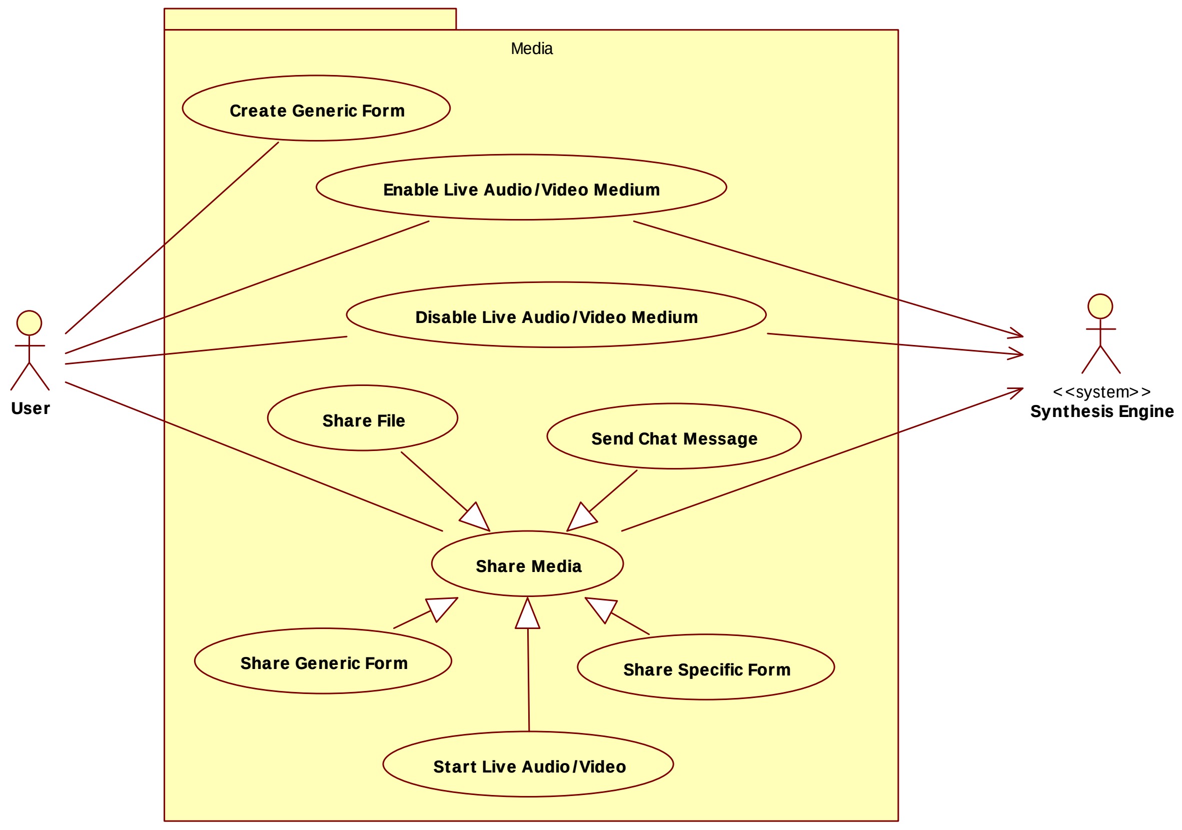


Figure 5 Media Package Use Cases

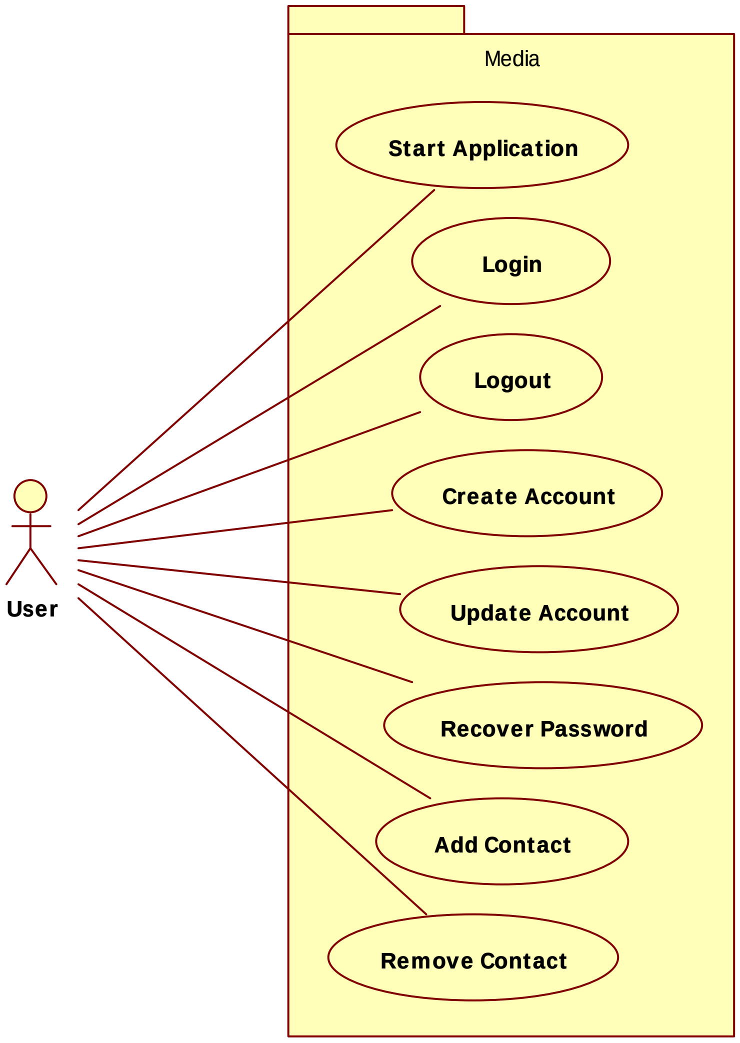


Figure 6 Account Package Use Cases

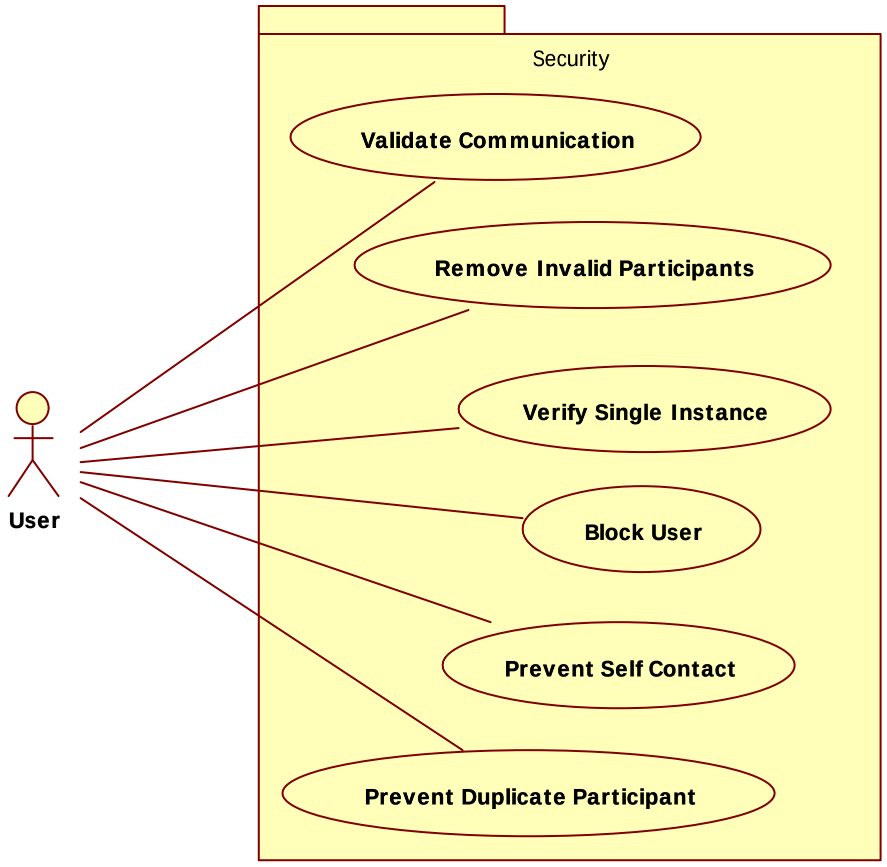


Figure 7 Security Package Use Cases

4.3 Requirements Analysis

The requirements analysis of the CVM GUI was achieved by describing eight usage scenarios of the system and documenting them using sequence diagrams in the analysis level. The scenarios defined for this activity are the following:

1. A user creates a new communication.
2. A user loads a communication schema.
3. A user adds a participant to a connection.
4. A user joins a connection.
5. A user shares a file in a connection.

6. A user starts streaming live audio/video.
7. A user saves a communication schema.
8. A user shares a generic form in a connection.

4.3.1 *Analysis Sequence Diagrams*

This subsection presents the analysis sequence diagrams created for each of the scenarios created during the analysis phase.

The following sequence diagram specifies interactions for the previously described scenario in which the user crates a new communication.

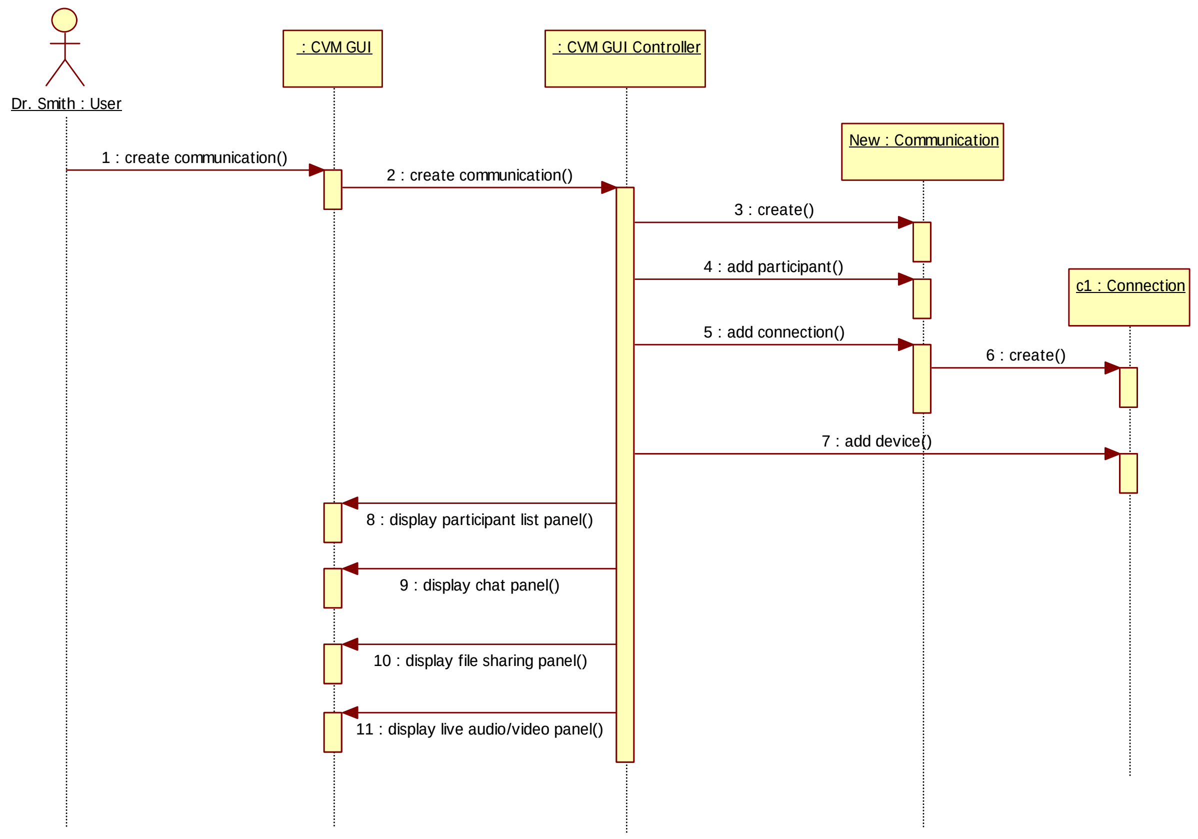


Figure 8 Object Diagram for Create Communication Scenario

The following sequence diagram specifies interactions for the previously described scenario in which the user loads a communication. Note how the X-CML is sent to the Synthesis Engine.

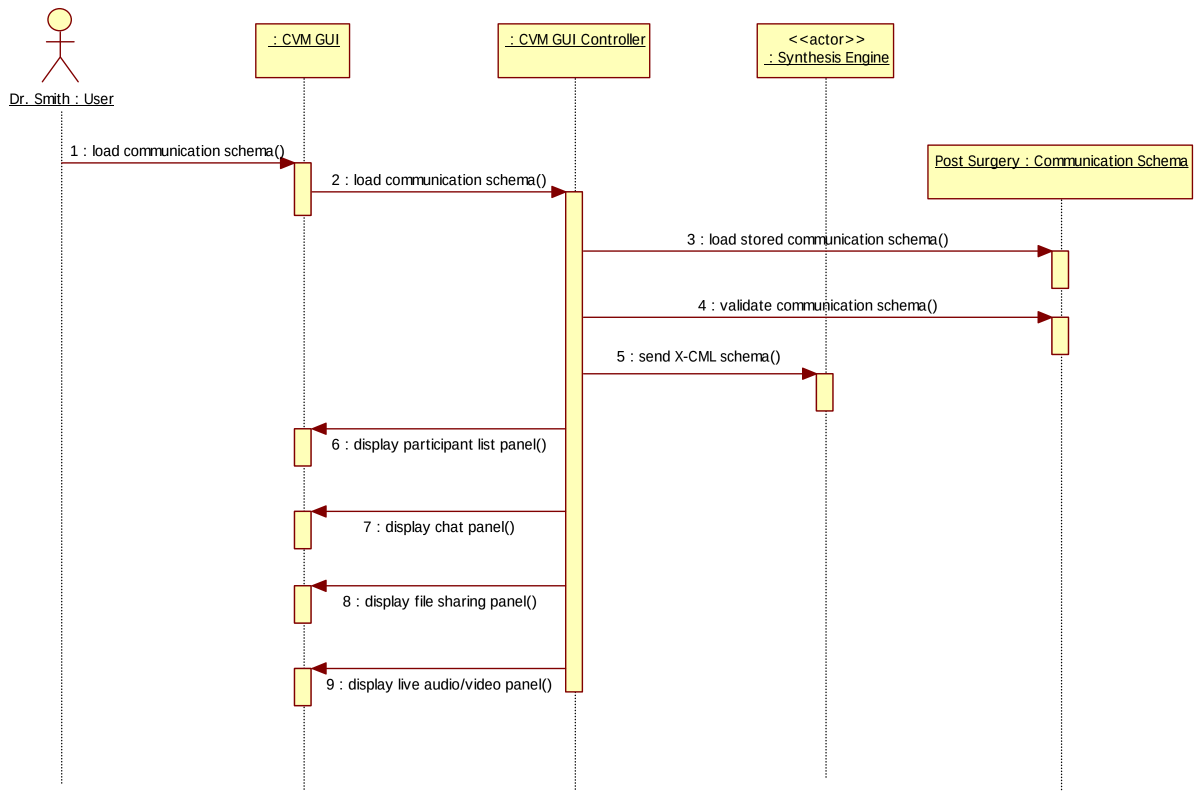

Dr. Smith : User
: CVM GUI
: CVM GUI Controller
<<actor>>
: Synthesis Engine
Post Surgery : Communication Schema
1 : load communication schema()
2 : load communication schema()
3 : load stored communication schema()
4 : validate communication schema()
5 : send X-CML schema()
6 : display participant list panel()
7 : display chat panel()
8 : display file sharing panel()
9 : display live audio/video panel()

Figure 9 Sequence Diagram for Load Communication Scenario

The following sequence diagram specifies interactions for the previously described scenario in which the user adds a participant to a communication. Note how the updated X-CML is sent to the Synthesis Engine.

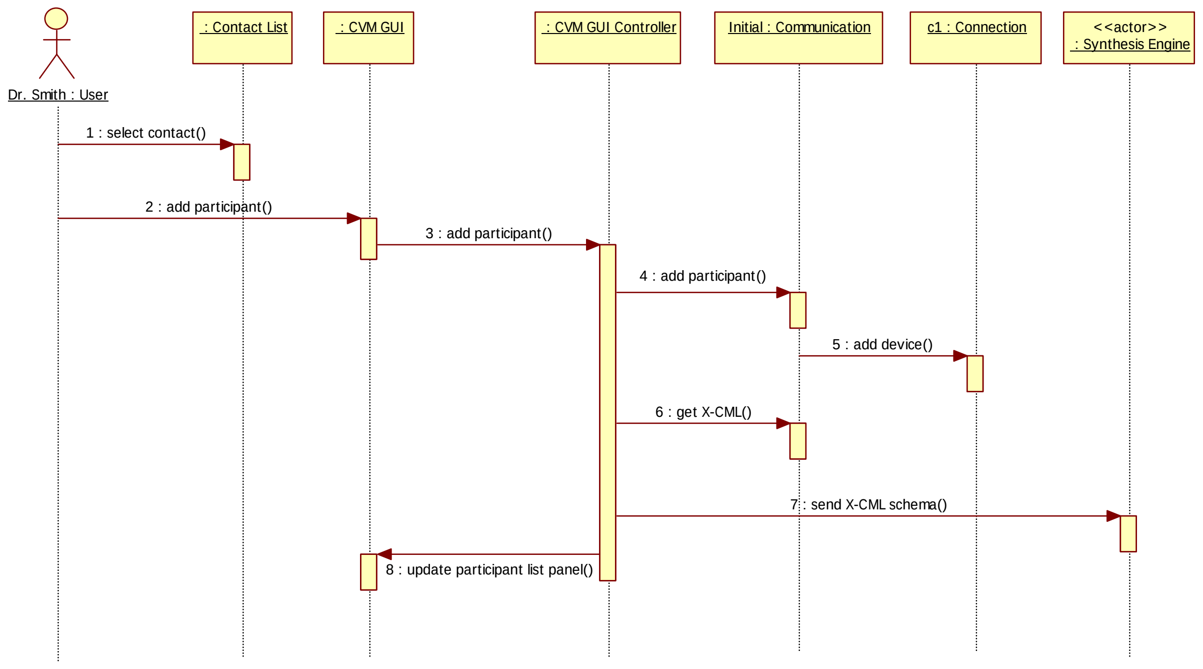


Figure 10 Sequence Diagram for Add Participant Scenario

The following sequence diagram specifies interactions for the previously described scenario in which the user joins a connection after being invited. Note how the synthesis engine provides the negotiated X-CML schema.

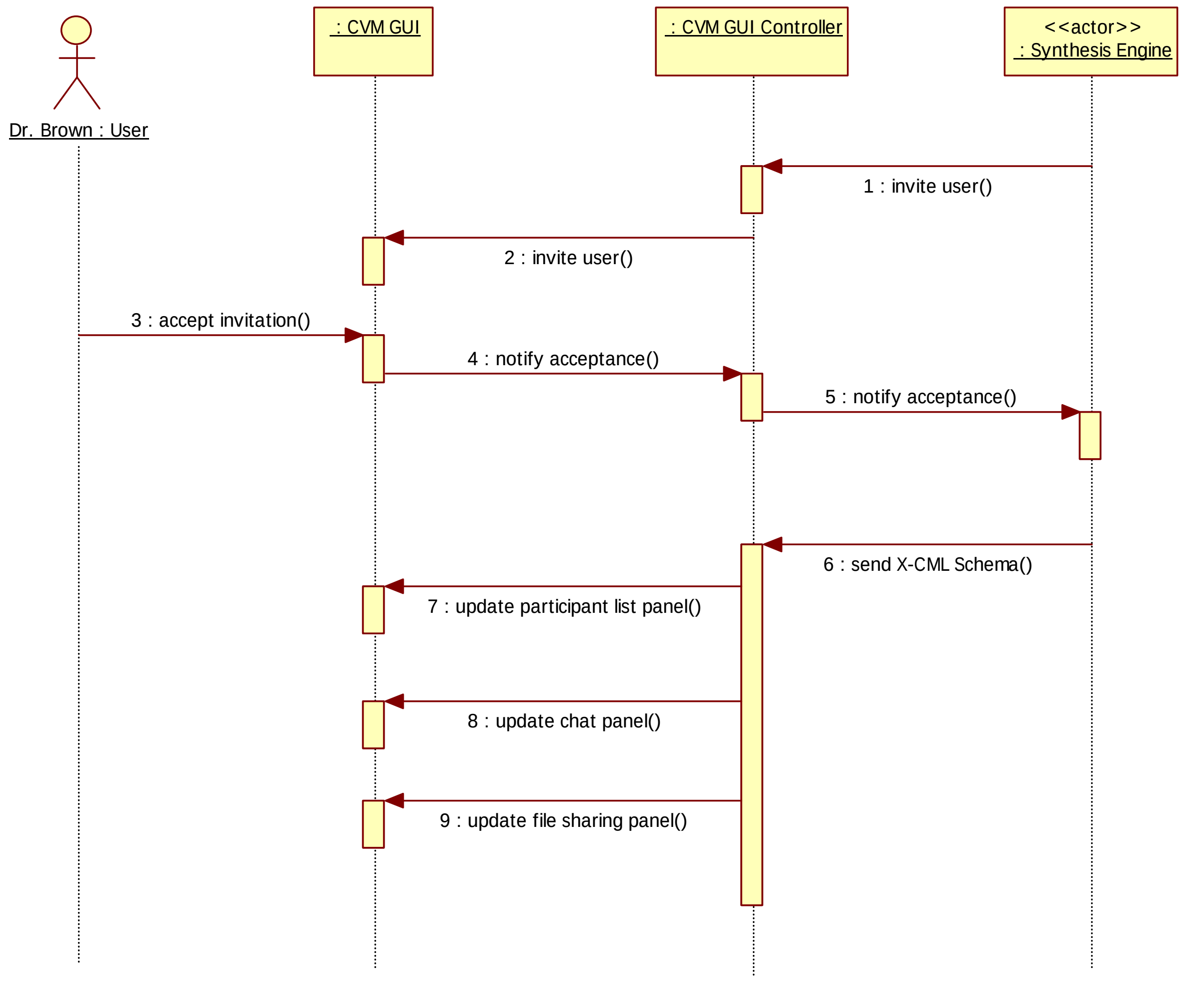


Figure 11 Sequence Diagram for Join Communication Scenario

The following sequence diagram specifies interactions for the previously described scenario in which the user shares a file in a connection.

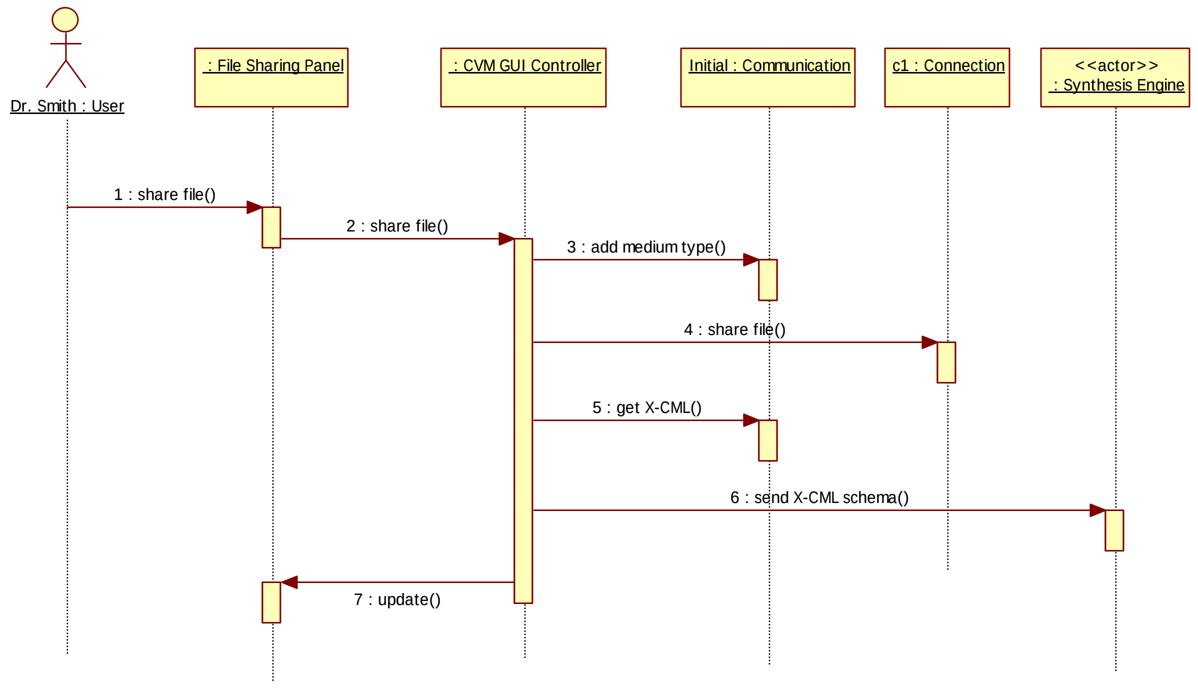


Figure 12 Sequence Diagram for Share File Scenario

The following sequence diagram specifies interactions for the previously described scenario in which the user starts live audio/video in a connection.

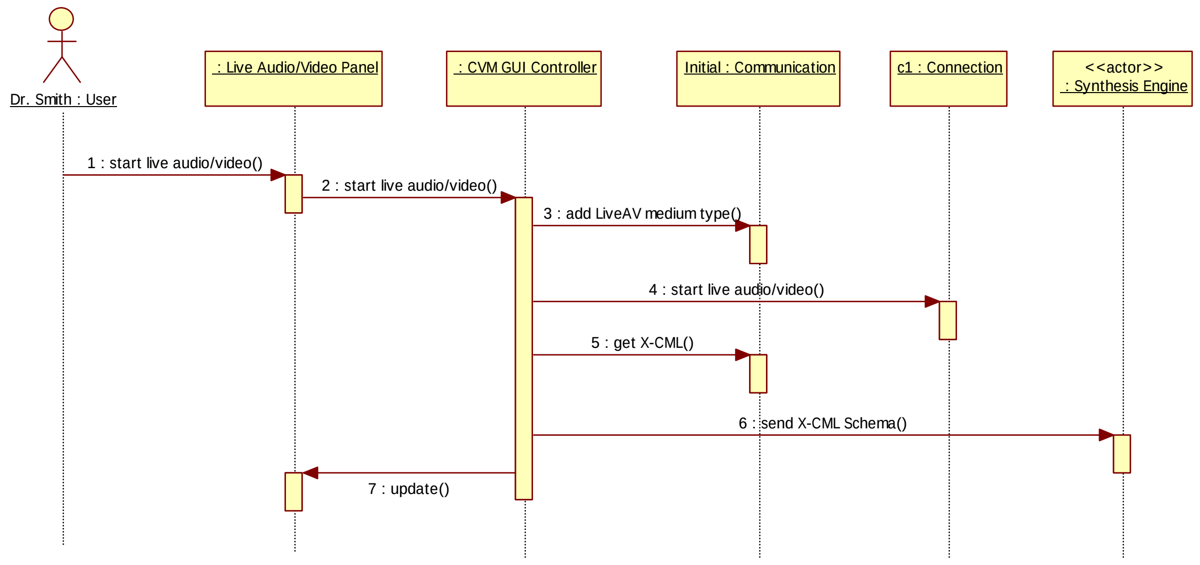


Figure 13 Sequence Diagram for Start Live Audio/Video Scenario

The following sequence diagram specifies interactions for the previously described scenario in which the user shares a generic form in a connection.

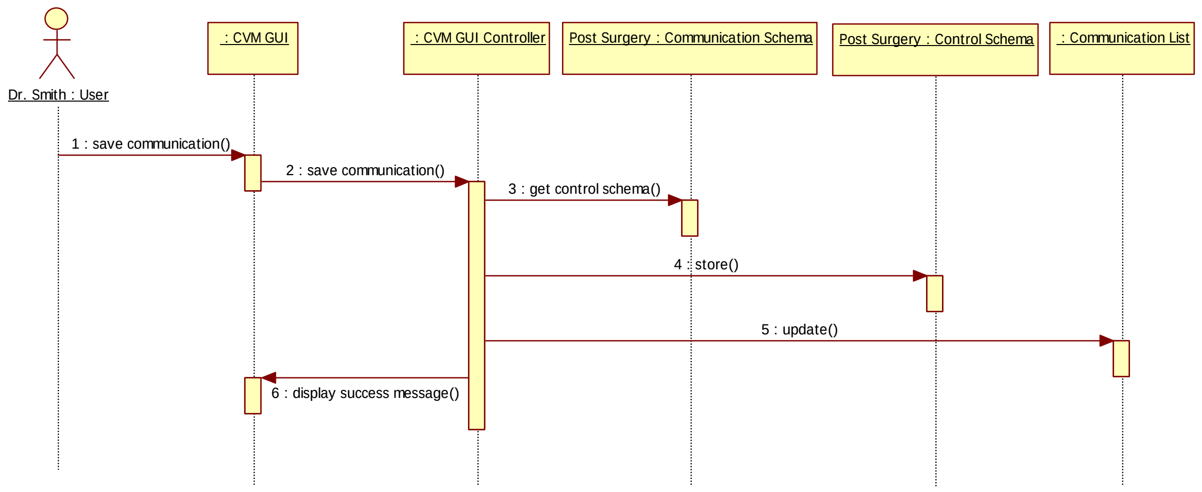


Figure 14 Sequence Diagram for Save Communication Scenario

5 SOFTWARE ARCHITECTURE

This section describes the results of the System Design for the CVM GUI. An architectural overview is first presented, identifying the subsystem decomposition. Next, a description of each subsystem and their interfaces is provided followed by the mapping of hardware to software. Finally the persistent data storage requirements are presented.

It should be noted, that the use cases that are being implemented for the CVM GUI are:

- Create a new communication service and start executing it.
- Save a communication.
- Load a communication.
- Add a participant to a connection.
- Join a connection.
- Leave a communication.
- Enable/disable live audio/video medium.
- Share media (chat messages and live audio/video)

5.1 OVERVIEW

Based on the non-functional requirements and on the specification of the CVM, we chose a layered architecture for the CVM GUI. This architecture is composed of two layers, the GUI and the UCI. The GUI subsystem provides the end-user with a means to create and execute communication services. The UCI provides the necessary interfaces for realizing communication services and interacts with the Synthesis Engine in order to achieve this goal. This system decomposition provides low coupling between the user interface and the application logic elements, allowing the UCI to be reused for future implementations of a user interface.

Based also on the specification of the CVM, we decided to use a repository architecture on the design of the CVM GUI. The central repository in this case is the Local Repository of the CVM. The Local Repository provides the functionality for storing communication services and sharing them with other subsystems of the CVM. In particular, it enables the interaction between the CVM GUI and the CVM Modeling Environment.

Figure 15 below depicts the dependencies between the subsystems that are part of the CVM GUI architecture. It also shows the dependencies between the CVM GUI and the Synthesis Engine of the CVM.

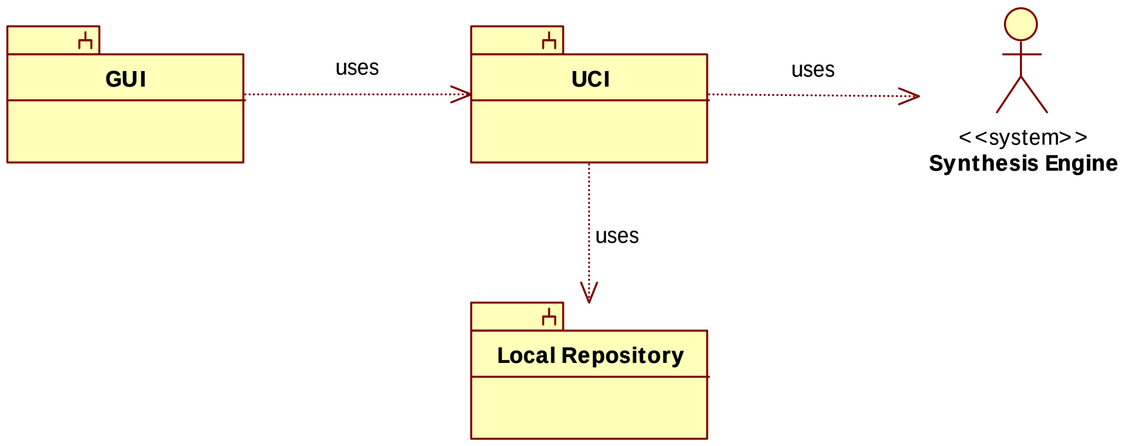


Figure 15 CVM GUI Architecture

5.2 Subsystem Decomposition

The following subsections describe each of the subsystems that are part of the CVM GUI.

5.2.1 GUI Subsystem

The GUI subsystem is the application used by the end-users of the CVM in order to achieve all the goals described in the Use Case Model. This component provides user interfaces that allow the user to:

- Create a new communication service and start executing it.
- Save a communication.
- Load a communication.
- Add a participant to a connection.
- Join a connection.
- Leave a communication.
- Create generic forms.

- Enable/disable mediums.
- Share media (files, chat messages, live audio/video)
- Share generic and specific forms.

It should be noted that the GUI subsystem does not implement any of the functionalities related with the execution of communication services, but delegates these tasks to the UCI.

Based on the non-functional requirements, the GUI subsystem should be implemented using the Eclipse SWT framework.

5.2.2 UCI

The UCI subsystem provides an interface that allows client applications to execute communication services. In order to realize its operations, the UCI uses the services provided by the Synthesis Engine.

The following is a list of the operations provided by the UCI:

- Retrieve the list of stored communications.
- Retrieve the list of stored generic forms.
- Retrieve the list of stored specific forms.
- Retrieve the list of available files.
- Store a form.
- Create a communication.
- Create a connection.
- Add a participant to a connection.
- Add a medium to a connection.
- Remove a medium from a connection.
- Share media on a connection.
- Share a form on a connection.
- Reply an invitation to join a communication.
- Leave a communication.
- Store a communication.

- Load a communication.

The UCI should consist of a set of components as depicted on Figure 16. The purpose of each component is as follows:

- X-CML Model: provides an object representation of the X-CML language.
- X-CML Parser: provides the functionalities for translating an X-CML communication schema into the object representation using the classes in the X-CML model. It is also responsible for conducting syntax checks on X-CML communication schemas.
- Schema Validation: this component performs the semantic validations required for X-CML communication schemas.
- UCI Engine: the UCI engine keeps the state of the communication service executing in the UCI and processes all the events from the user interface and the Synthesis Engine.
- Model: provides an abstract representation of a communication service. The purpose of the model is decoupling the applications using the services provided by the UCI from the syntax of X-CML.
- UCI: the UCI Interface provides the operations previously listed. The operations of the UCI Interface use the types defined in the Model component for the specification of the input values and return values.

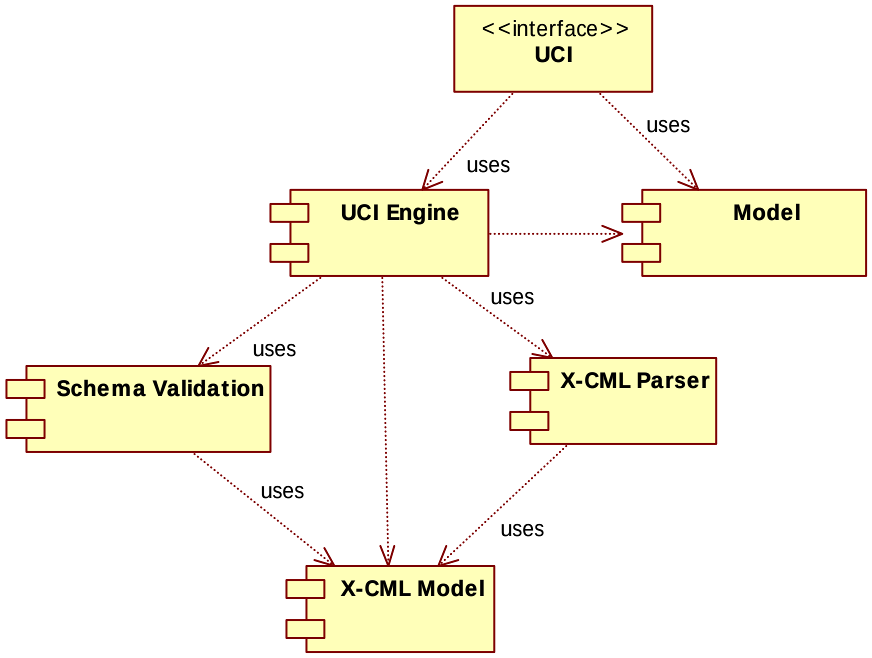


Figure 16 UCI Components

5.2.3 *Local Repository*

The local repository subsystem provides access to the communication related data that has been persisted. In particular it provides the following services:

- Retrieve the list of stored communication files.
- Store a communication file.
- Load a communication file.
- Retrieve the list of stored generic form files.
- Retrieve the list of stored specific form files.
- Retrieve the list the available files.

5.3 Hardware and Software Mapping

The CVM GUI is a java SWT application and should be deployed on the end-user's device. It is distributed as an executable jar file that automatically builds the structure of the local repository using the device's file system. This deployment arrangement is illustrated in Figure 17.

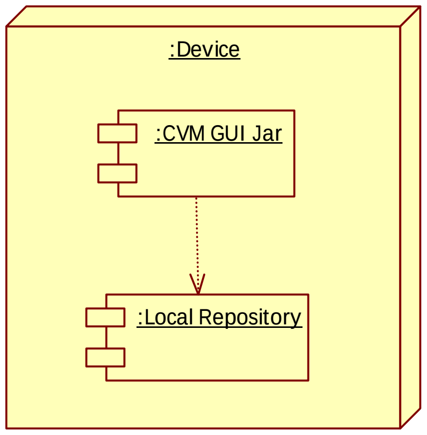


Figure 17 CVM GUI Deployment

The hardware requirements for the CVM GUI are the following:

- 1 GHz 32-bit (x86) processor or similar.
- 1 GB of system memory.
- 1 GB of available storage space.
- Internet Access.

The software requirements for the CVM GUI are the following:

- Operating system supported by Java.
- Java SDK 6.

An example deployment configuration that enables the communication between two users is illustrated in Figure 18. The specifications of PC1 and PDA1 are the following:

PC1:

- 1 GHz 32-bit (x86) processor or similar.
- 1 GB of system memory.
- 1 GB of available storage space.
- Internet Access.

PDA1:

- Pocket PC with Java Support
- 64 MB of system memory.
- 256 MB of available storage space.
- Internet Access.

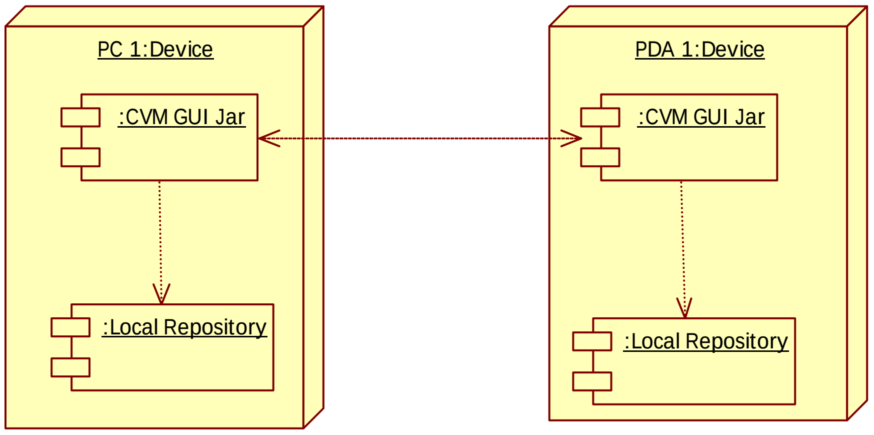


Figure 18 CVM GUI Deployment Example 1

If an end-user is executing both the CVM GUI and the CVM Modeling environment the deployment configuration will looks as illustrated in Figure 19.

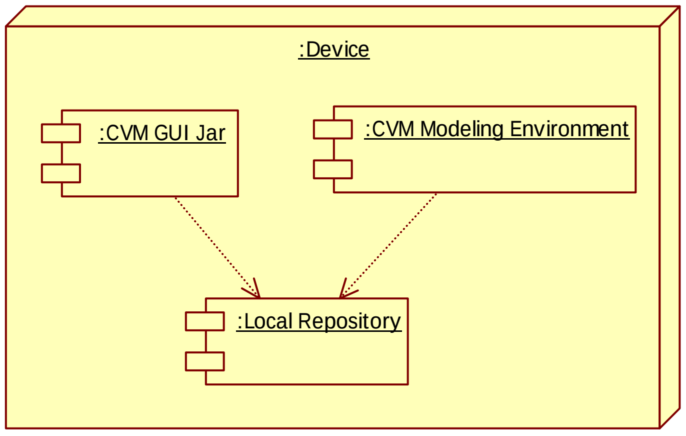


Figure 19 CVM GUI Deployment Example 2

5.4 Persistent Data Management

The persistent data is stored on the Local Repository. The repository is implemented as a set of operating system files and folders. The following information needs to be stored in the repository:

- Communication schemas.
- Generic Forms.
- Specific Forms.
- Files that can be shared in a communication.

Each of these should be stored as an individual file system file and separated in folders, one folder for each type of file. Each type of file is independent, that is, there is no relationship between them.

6 Object Design

This chapter introduces the Detailed Design of UCI and GUI components. The first section displays minimal class diagrams along with brief explanations of each class. The next section depicts state machines for both UCI and GUI subsystems, followed by sequence diagram that illustrates object interaction of our stated scenarios. The final section goes into more detailed explanation of each class.

6.1 Overview

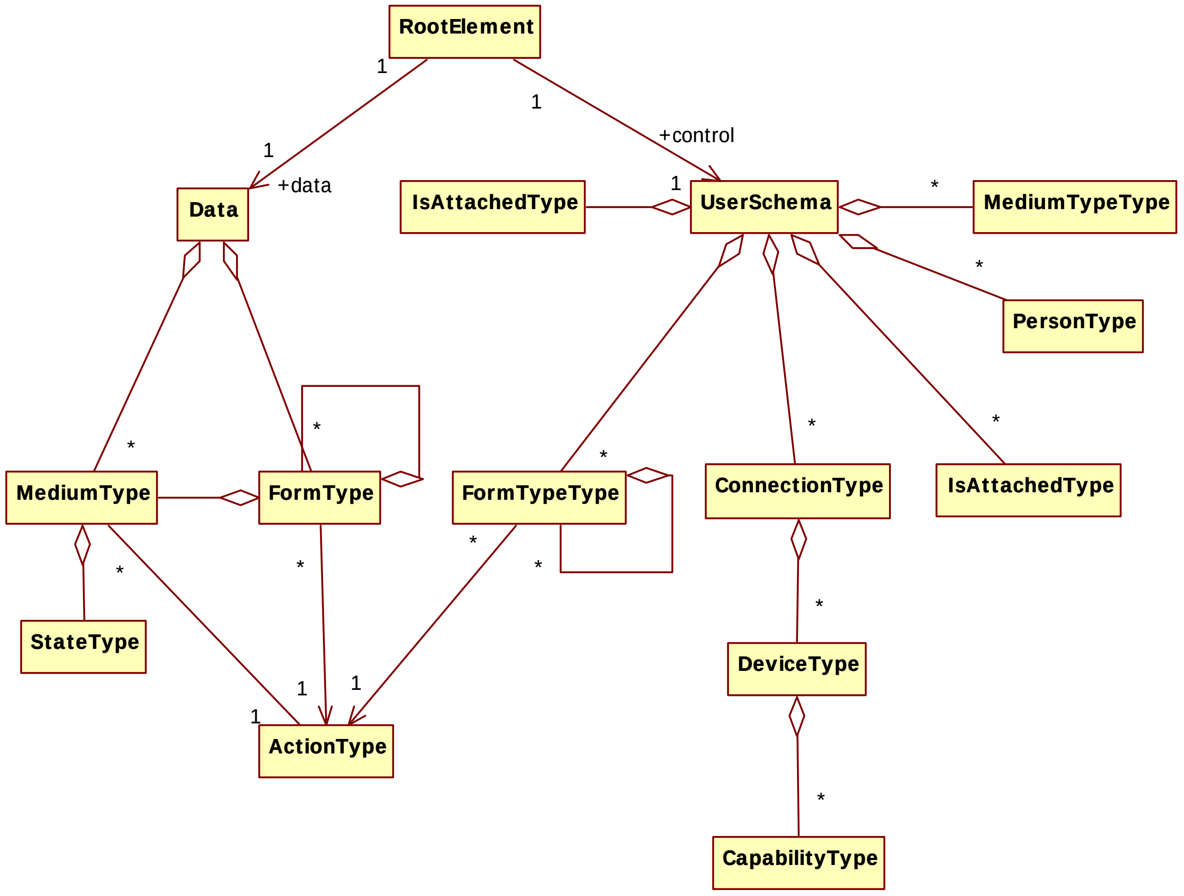


Figure 20 XCML Minimal Class Diagram

- ValidationError – This class represent a validation error.

- XCMLObjectFactory – This class adds syntactic and XML semantic error detection/handling with the XCMLValidationEventHandler.
- XCMLValidationEventHandler – This class handles syntactic and XML semantic errors while marshaling or un-marshaling XCML object trees.
- XCMLVisitor(interface) – This class navigates XCML object trees.
- ObjectFactory – A class generated by JAXB based on the XSD to create XCML object trees.
- RootElement – This class wraps the UserSchema and Data node types.
- ActionType – A class generated by JAXB based on the XSD to create XCML object trees.
- CapabilityType – A class generated by JAXB based on the XSD to create XCML object trees.
- ConnectionType – A class generated by JAXB based on the XSD to create XCML object trees.
- Data – A class generated by JAXB based on the XSD to create XCML object trees.
- DeviceType – A class generated by JAXB based on the XSD to create XCML object trees.
- FormType – A class generated by JAXB based on the XSD to create XCML object trees.
- FormTypeType – A class generated by JAXB based on the XSD to create XCML object trees.
- IsAttachedType – A class generated by JAXB based on the XSD to create XCML object trees.
- MediumType – A class generated by JAXB based on the XSD to create XCML object trees.
- MediumTypeType – A class generated by JAXB based on the XSD to create XCML object trees.
- PersonType – A class generated by JAXB based on the XSD to create XCML object trees.
- StateType – A class generated by JAXB based on the XSD to create XCML object trees.
- UserSchema – A class generated by JAXB based on the XSD to create XCML object trees.

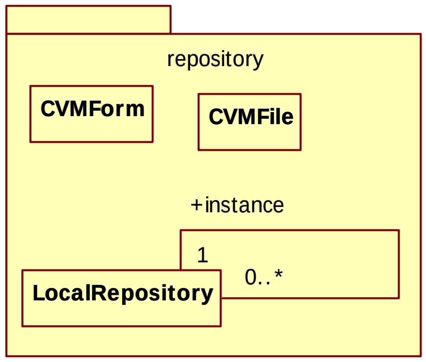


Figure 21 Repository Minimal Class Diagram

- LocalRepository – Stores Communications, Files, and Forms on disk.
- CVMForm – A stored form type.
- CVMFile – A stored file type.

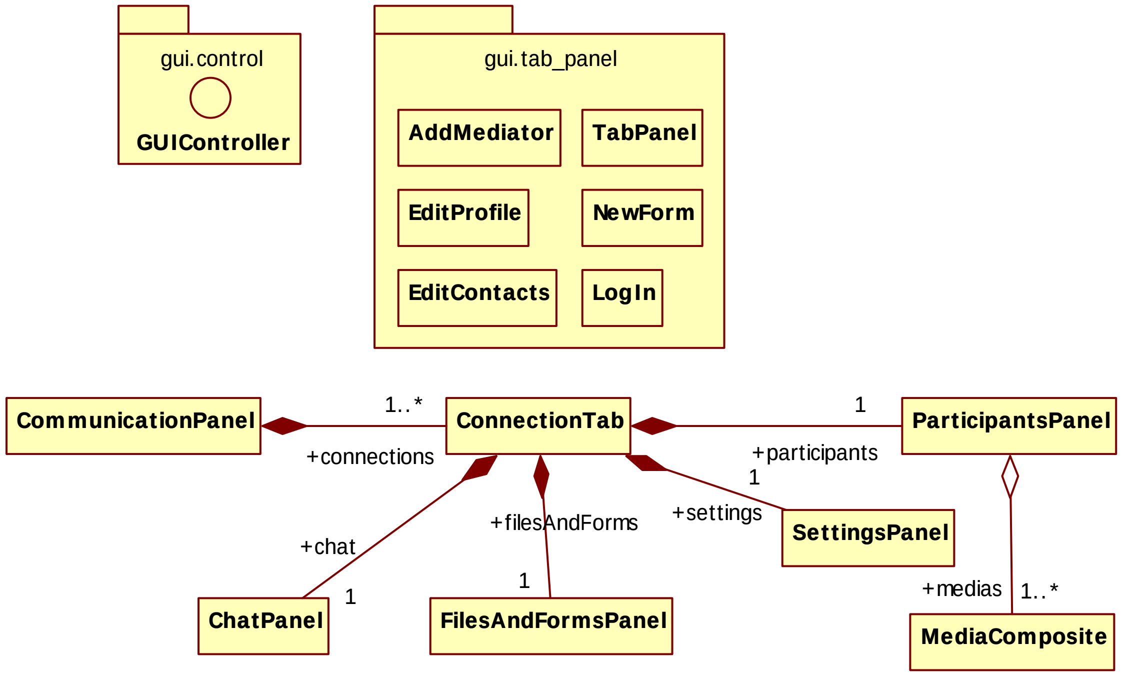


Figure 22 GUI Minimal Class Diagram

- ChatPanel - A traditional chat control panel: history, new text, send button.
- Communication - This is the GUI class that represents the communication window.
- ConnectionTab - A tab in the communication window. It represents a connection.
- FileAndFormsPanel - A panel that shows a table with the shared files and forms.
- MediaComposite - Control that represents a participant on a connection.
- ParticipantsPanel - Panel that shows a group of participants.
- SettingsPanel – A panel with basic controls for managing audio and video settings: audio/mic volume, audio/video enabling or disabling.
- GuiController (interface) - This is the event handling class for the GUI component.
- GuiControllerImp – This is the implementation of the GuiController interface.
- AddMediator - This class creates Add Mediator dialog box.
- EditContacts - This class creates Edit Contacts dialog box.
- EditProfile - This class creates Edit Profile dialog box.
- LogIn - This class is the log in pane for CVM.
- NewForm - This class creates New Form dialog box.
- TabPanel - This class is the main application window composed of tabs.

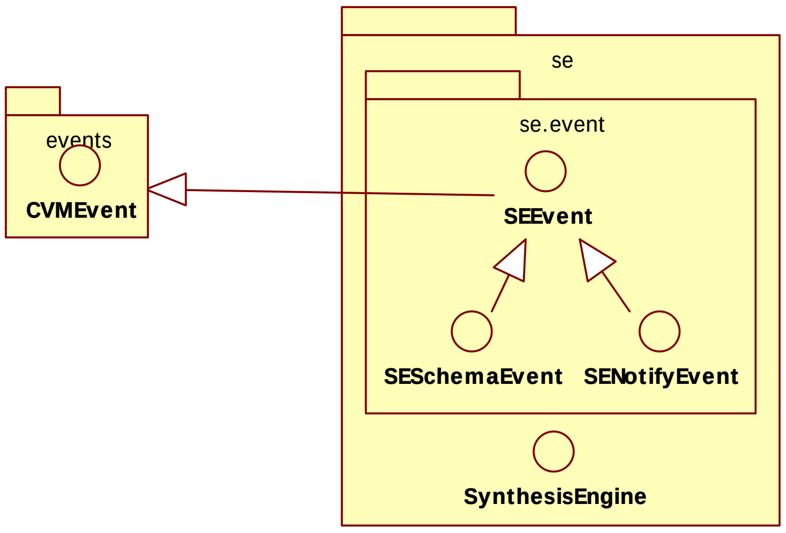


Figure 23 Synthesis Engine Class Diagram

- SynthesisEngine – The façade exposes functions exchange XCML schemas and events with the UCI.
- SEEvent – This is the root interface for all events that originate from the Synthesis Engine.
- SENotifyEvent – This is a particular SE Event that is used to present notifications/alerts from the SE.
- SESchemaEvent – This is a particular SE Event that is used to notify the UCI of a control schema updates and new data schemas.
- SynthesisEngineImpl – This is a testing stub that mimics the required functionality presented in the SE façade.

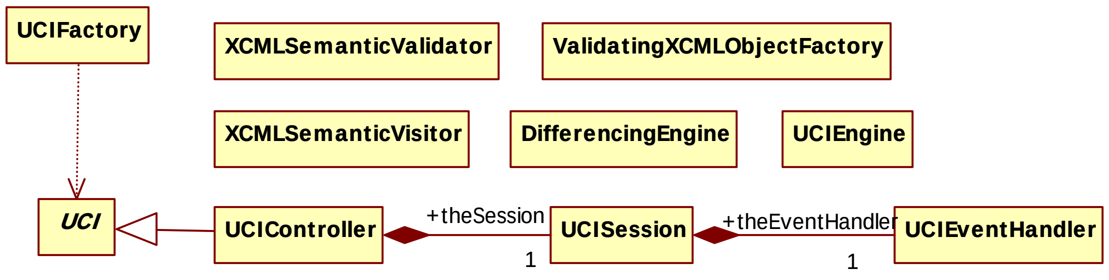


Figure 24 UCI.IMPL Minimal Class Diagram

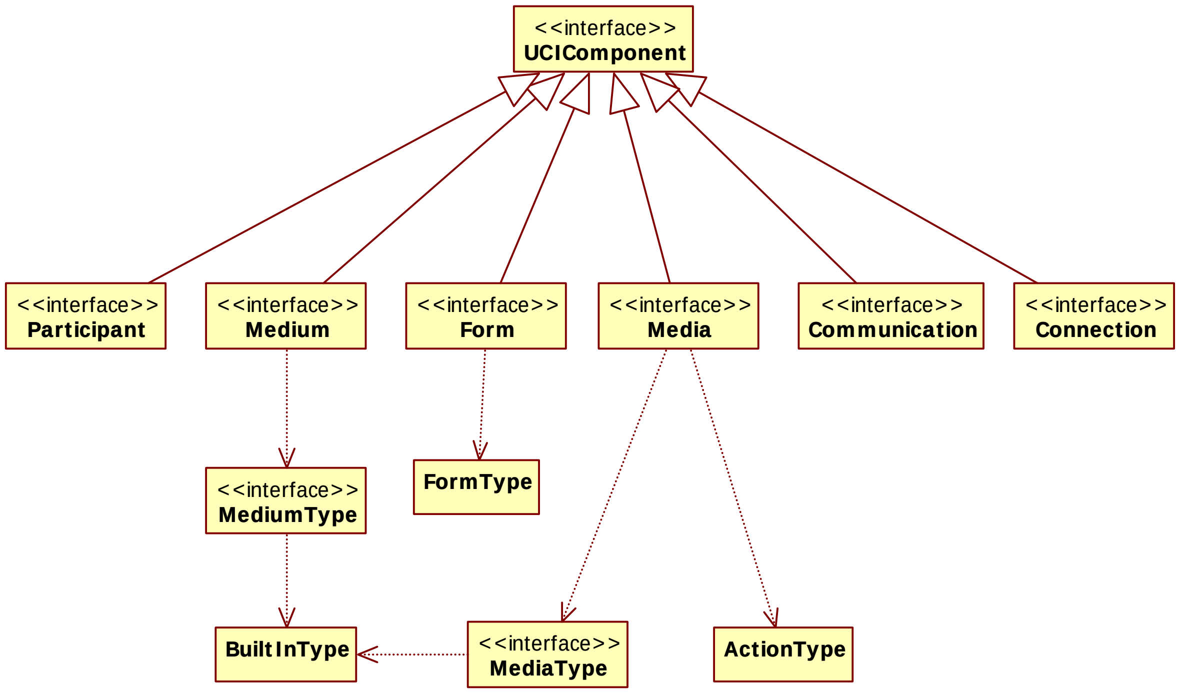


Figure 25 UCI.COMPONENT Minimal Class Diagram

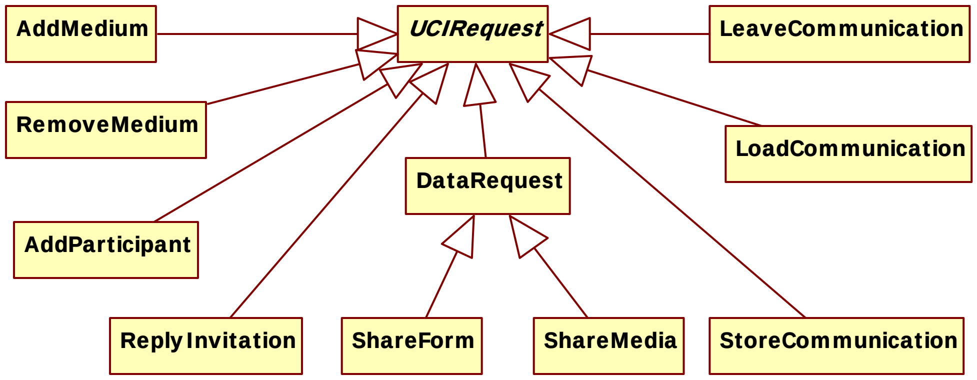


Figure 26 UCI.IMPL.REQUESTS Minimal Class Diagram

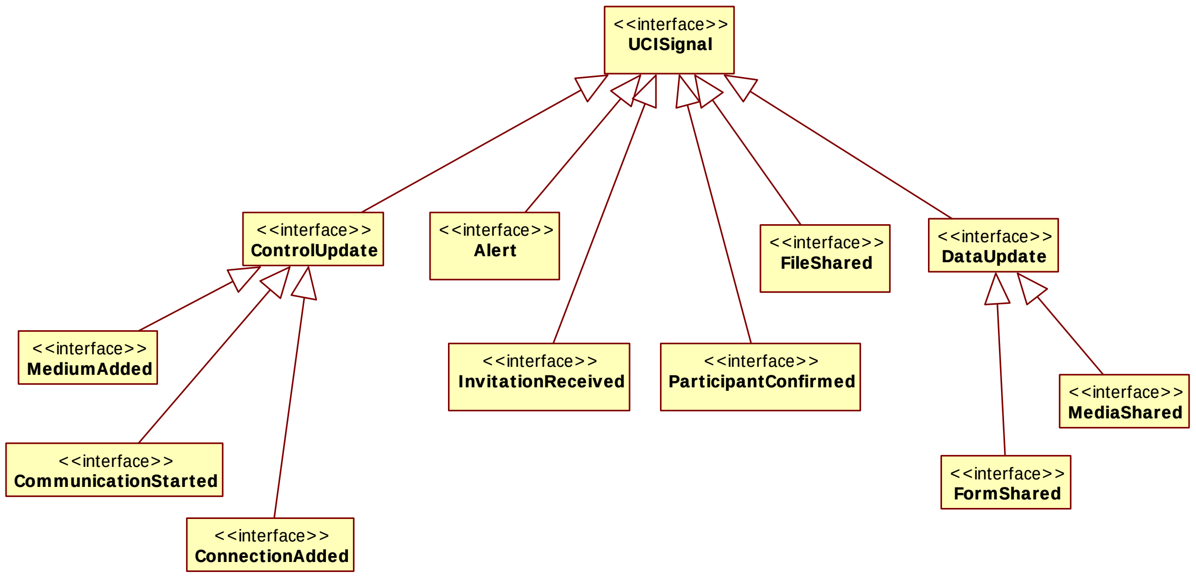


Figure 27 UCI.SIGNAL Minimal Class Diagram

- UCI - It exposes all the necessary methods for the SE and GUI to communicate with the UCI.
- UCIFactory – This is a factory for the UCI.
- ActionType - This enum represents the possible action types in XCML.
- BuiltInType - This enum defines the supported built in types. These types are the basis for medium and media types
- UCIRequest – This is the base interface for UCI events.
- DataRequest (interface) - This is the base interface for UCI data requests.
- Alert - Allow the UCI to present alerts such as errors to the GUI.
- InvitationReceived - Allows UCI to prompt GUI to ask user if they wish to join a communication.
- FileShared - Allows UCI to respond to the OpenSharedFile request.
- UCISignal - This is the base interface for UCI signal events.
- AlertEnum - The types of alert that can be sent to the GUI.
- ParticipantConfirmed - Allow the UCI to notify the GUI when a newly added participant accepts an invitation.
- UCIComponent - This interface is a base interface for all UCIRequestcomponents.

- Communication – A UCIRequestcomponent that allows the GUI and UCI to create and delete a connection.
- Connection - A UCIRequestcomponent that allows the GUI to add a connection to a communication.
- Form - A UCIRequestcomponent that allows the GUI to share a form in a connection.
- FormType - A UCIRequestcomponent that allows the GUI to build new Form Types and add them to a communication.
- Media - A UCIRequestcomponent that allows the UCI/GUI to talk about instances of media.
- MediaType - A UCIRequestcomponent that allows the GUI/UCI to talk about media types.
- Medium - A UCIRequestcomponent that allows the GUI/UCI to add/remove mediums to a connection/communication.
- MediumType - A UCIRequestcomponent that allows the UCI/GUI to talk about user defined and built in medium types.
- Participant - A UCIRequestcomponent that allows the GUI/UCI to add/update/remove people to connections.
- AudioFile - This class represents an audio file.
- AudioVideoFile - This class represents an audio video file.
- BinaryFile - This class represents a binary file.
- TextFile - This class represents a Text file.
- TextMessage - This class represents a Text message.
- VideoFile - This class represents a Video file.
- LiveAudio - This class represents live audio.
- LiveAudioVideo - This class represents live audio video.
- LiveVideo - This class represents a live video.
- UCIException - Class to wrap all errors that might happen in the UCI.
- DifferencingEngine - A class that is able to extract a series of UCIEvents from two control schemas. The series of UCIEvents represent the difference between the two schema.
- UCIController – This is the class that manages the main control flow for the UCI.
- UCIEngine - This class does the work of processing the events and schemas from the upper and lower layers the UCI is attached to.

- UCIEventHandler – This class is a thread for the UCI which initiates the sending and receiving of events between the upper and lower layers the UCI is attached to.
- UCISession - This class stores all the state associated with a communication.
- ValidatingXCMLObjectFactory -. This class add complex semantic validation by decorating an XCMLObjectFactory.
- XCMLSemanticVistor (interface) - This interface extends the generic XCMLVisitor for the XCML validation vistor to implement.
- XCMLSemanticValidator – This class implements the XCMLSemanticVistor interface and validates a XCML object tree.
- CommunicationImpl - This class is an implementation of a Communication.
- ConnectionImpl - This class is an implementation of a Connection.
- FormImpl - This class is an implementation of a Form.
- FormTypeImpl - This class is an implementation of a FormType.
- MediaImpl - This class is an implementation of a Media.
- MediaTypeImpl - This class is an implementation of a MediaType.
- MediumImpl - This class is an implementation of a Medium.
- MediumTypeImpl - This class is an implementation of a MediumType.
- ParticipantsImpl- This class is an implementation of a Participant.
- AddMedium – A UCI Request that allows the UCI/GUI to start/add a medium (start video/audio/etc).
- AddParticipant - A UCI Request that allows the UCI/GUI to add a participant.
- LeaveCommunication - A UCI Request that allows the GUI to leave/close a communication.
- LoadCommunication - A UCI Request that allows GUI to request the loading of a saved communication.
- RemoveMedium - A UCI Request that allows the UCI/GUI to stop/remove a medium (stop video/audio/etc).
- ReplyInvitation - A UCI Request that allows the GUI to send a invitation response to the UCI.
- ShareForm - A UCI Request that allows the GUI/UCI to share forms.
- ShareMedia - A UCI Request that allows the GUI/UCI to share media.
- StoreCommunication - A UCI Request that allows llows the GUI to request that the UCI save the communication.

6.2 State Machine

The following two UML Statechart diagrams are for the implementation of the GUIController class of the GUI subsystem.

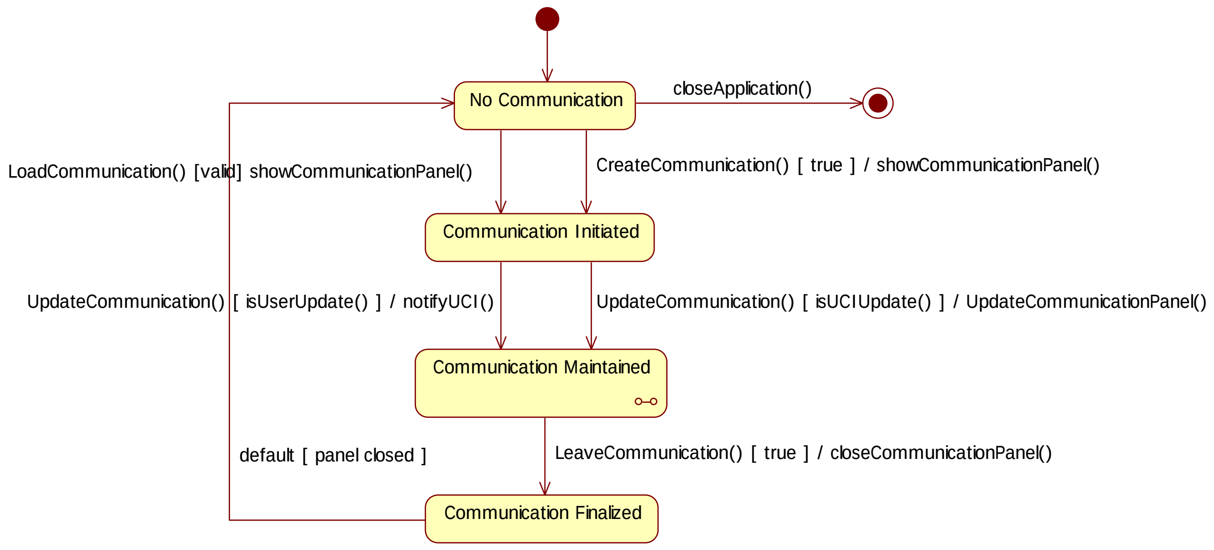

No Communication
closeApplication()
LoadCommunication() [valid] showCommunicationPanel()
CreateCommunication() [true] / showCommunicationPanel()
Communication Initiated
UpdateCommunication() [isUserUpdate()] / notifyUCI()
UpdateCommunication() [isUCIUpdate()] / UpdateCommunicationPanel()
Communication Maintained
default [panel closed]
LeaveCommunication() [true] / closeCommunicationPanel()
Communication Finalized

Figure 28 GUI Top Level State Machine

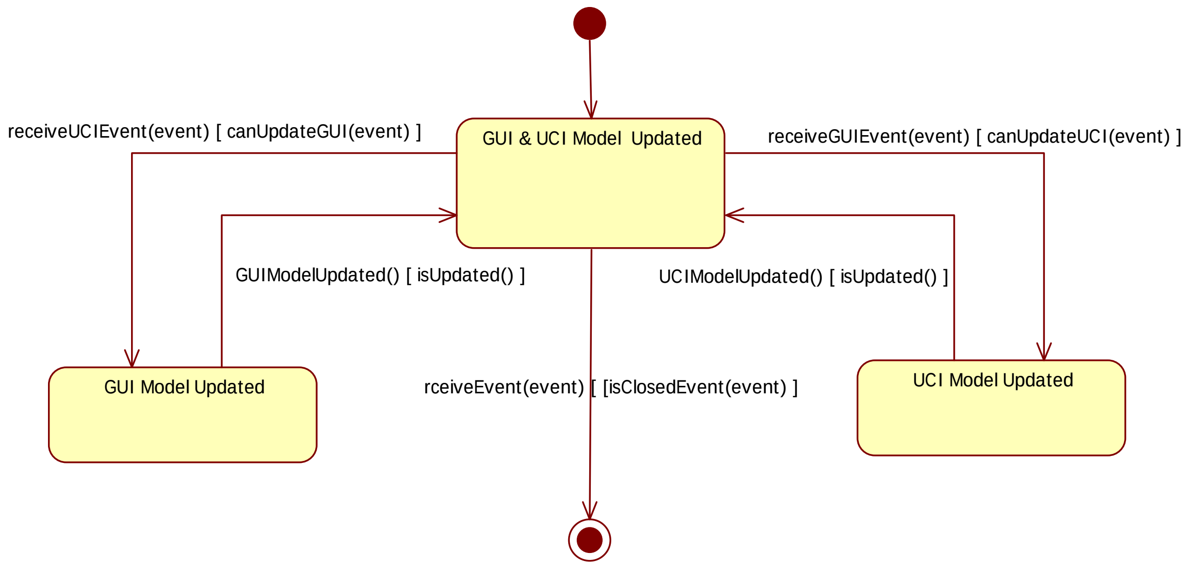

receiveUCIEvent(event) [canUpdateGUI(event)]
GUI & UCI Model Updated
receiveGUIEvent(event) [canUpdateUCI(event)]
GUIModelUpdated() [isUpdated()]
UCIModelUpdated() [isUpdated()]
GUI Model Updated
rceiveEvent(event) [isClosedEvent(event)]
UCI Model Updated

Figure 29 Communication Maintained Sub-Machine

The following two UML Statechart diagrams are for the UCIController class of the UCI subsystem.

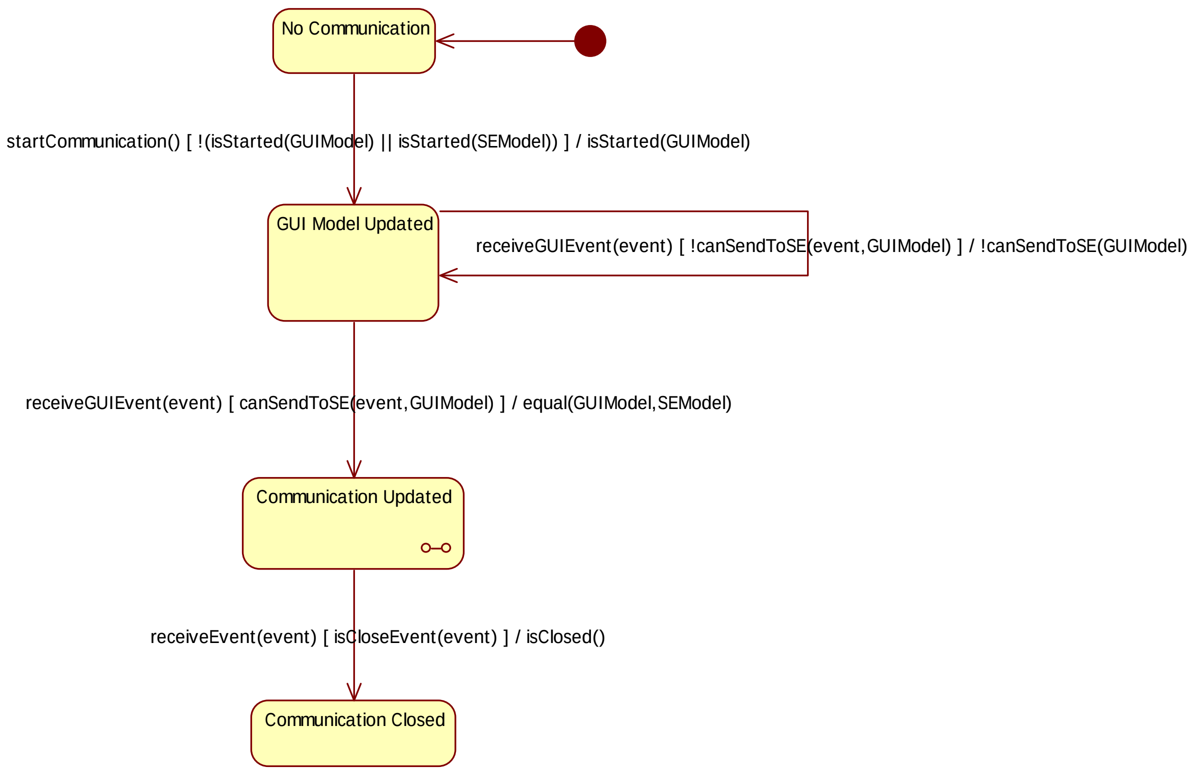

No Communication
startCommunication() [!(isStarted(GUIModel) || isStarted(SEModel))] / isStarted(GUIModel)
GUI Model Updated
receiveGUIEvent(event) [!canSendToSE(event,GUIModel)] / !canSendToSE(GUIModel)
receiveGUIEvent(event) [canSendToSE(event,GUIModel)] / equal(GUIModel,SEModel)
Communication Updated
receiveEvent(event) [isCloseEvent(event)] / isClosed()
Communication Closed

Figure 30 Top Level UCI State Machine

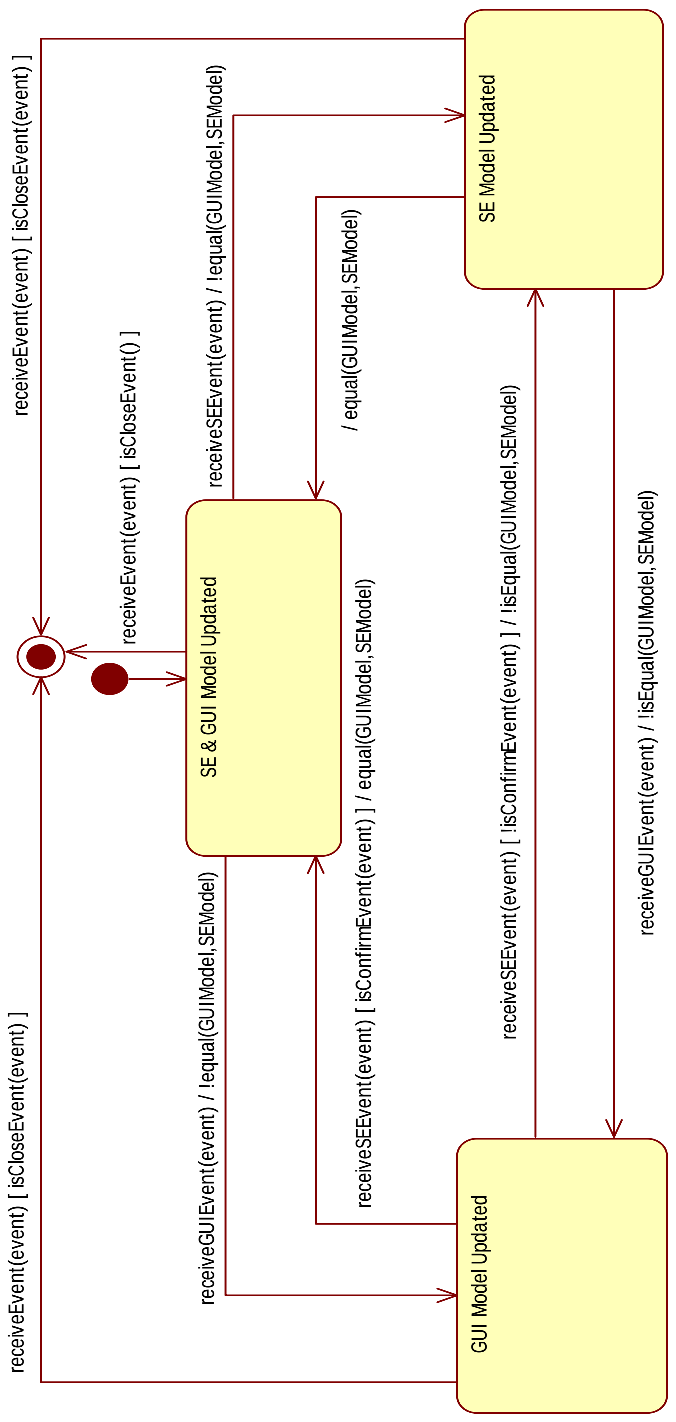

receiveEvent(event) [isCloseEvent(event)]
receiveEvent(event) [isCloseEvent()]
receiveSEEvent(event) / !equal(GUIModel,SEModel)
/ equal(GUIModel,SEModel)
SE Model Updated
receiveSEEvent(event) [!isConfirmEvent(event)] / !isEqual(GUIModel,SEModel)
receiveGUIEvent(event) / !isEqual(GUIModel,SEModel)
SE & GUI Model Updated
receiveEvent(event) [isCloseEvent(event)]
receiveGUIEvent(event) / !equal(GUIModel,SEModel)
receiveSEEvent(event) [isConfirmEvent(event)] / equal(GUIModel,SEModel)
GUI Model Updated

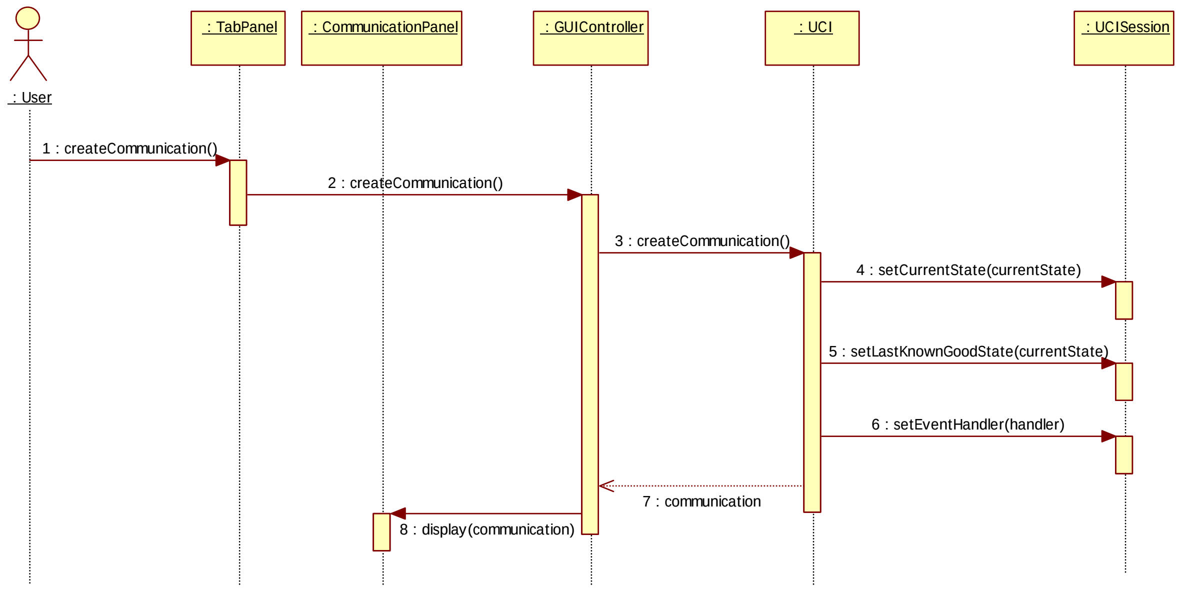
Figure 31 UCI Communication Updated Sub-Machine

6.3 Object Interaction

This subsection describes the object interaction in the CVMGUI by providing a set of sequence diagrams.

6.3.1 Create Communication

The following sequence diagram specifies interactions between objects that occur when a new communication is created.



: User
: TabPanel
: CommunicationPanel
: GUIController
: UCI
: UCISession
1 : createCommunication()
2 : createCommunication()
3 : createCommunication()
4 : setCurrentState(currentState)
5 : setLastKnownGoodState(currentState)
6 : setEventHandler(handler)
7 : communication
8 : display(communication)

Figure 32 Create Communication Sequence

6.3.2 End-User Updates Communication Schema

All control schema and data schema updates initiated by the end-user have been designed to use a similar sequence of interactions. The goal of this design is to provide asynchronous invocation for this operation, achieving high responsiveness for the user interface.

The following sequence diagram illustrates how a Command pattern is used to defer the execution of the UI requests. This diagram describes the object interaction for adding a participant *p* to a connection c on a session s. This sequence can be generalized for any schema update from the user interface, by replacing the request with the appropriate command.

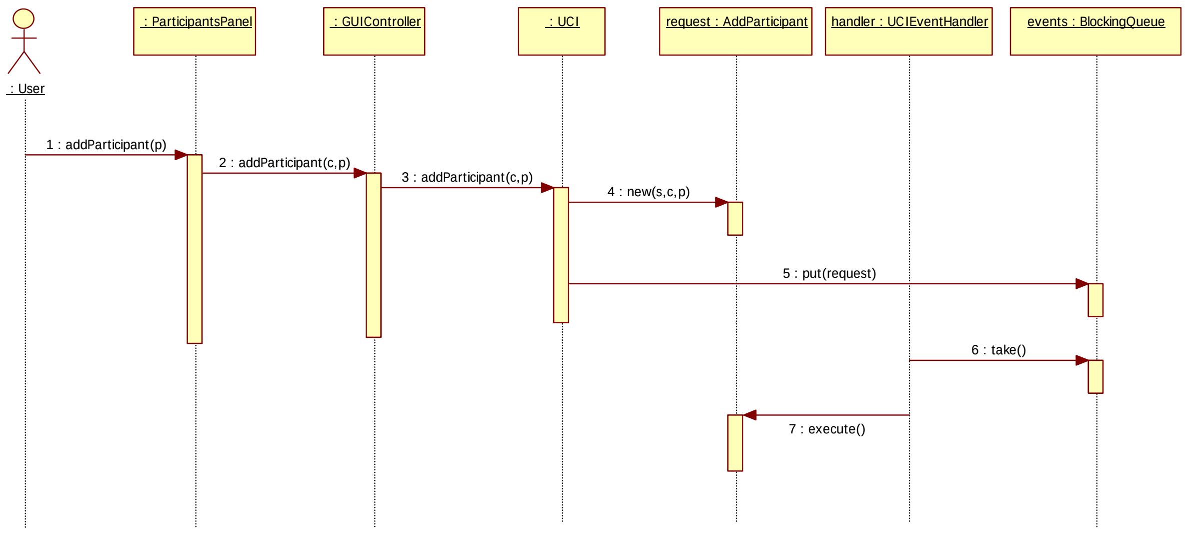


Figure 33 Add participant to a connection sequence

6.3.3 END-USER UPDATES CONTROL SCHEMA

The sequence of events occurring when the *execute* method of a command is called is very similar for all control schema updates. The following sequence diagram illustrate how the *execute* operation is handled for adding a participant to a connection. This sequence can be easily generalized for all control schema updates by simply changing the updates applied to the current state of the control schema.

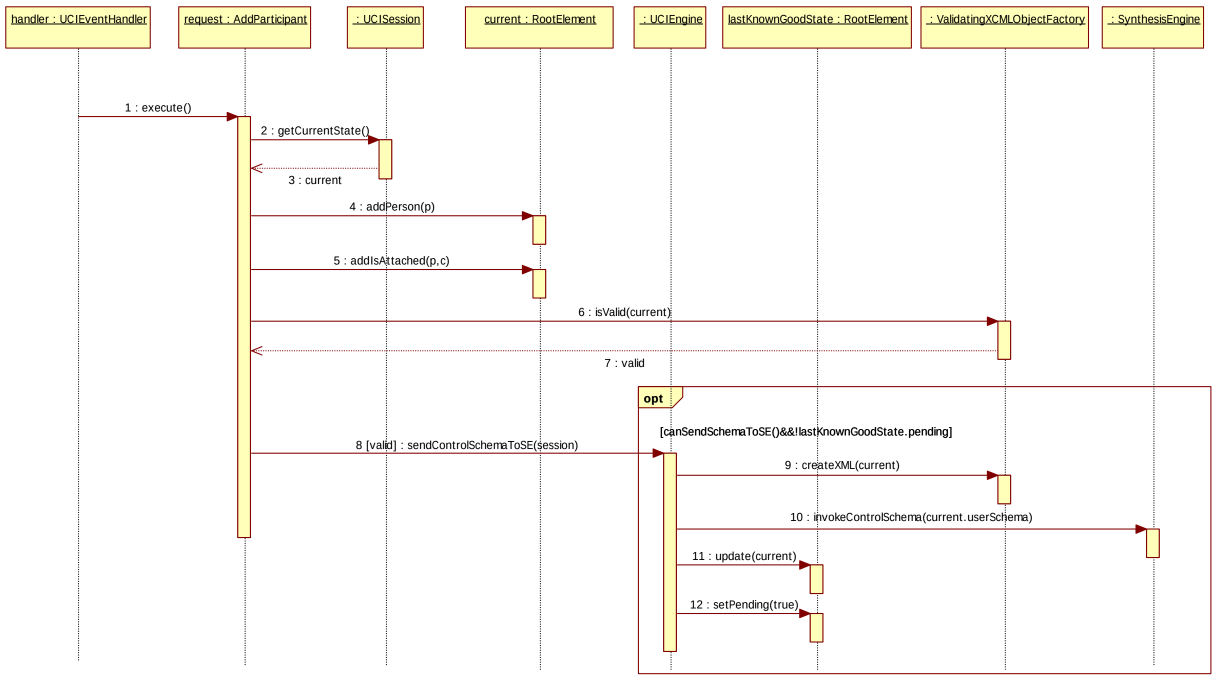


Figure 34 Add participant to a connection UCI Event Handler sequence

6.3.4 END-USER UPDATES DATA SCHEMA

The sequence of events occurring when the *execute* method of a command is called is also very similar for all data schema updates. The following sequence diagram illustrates the object interactions for sharing media in a connection. The process for sharing a form is analogous.

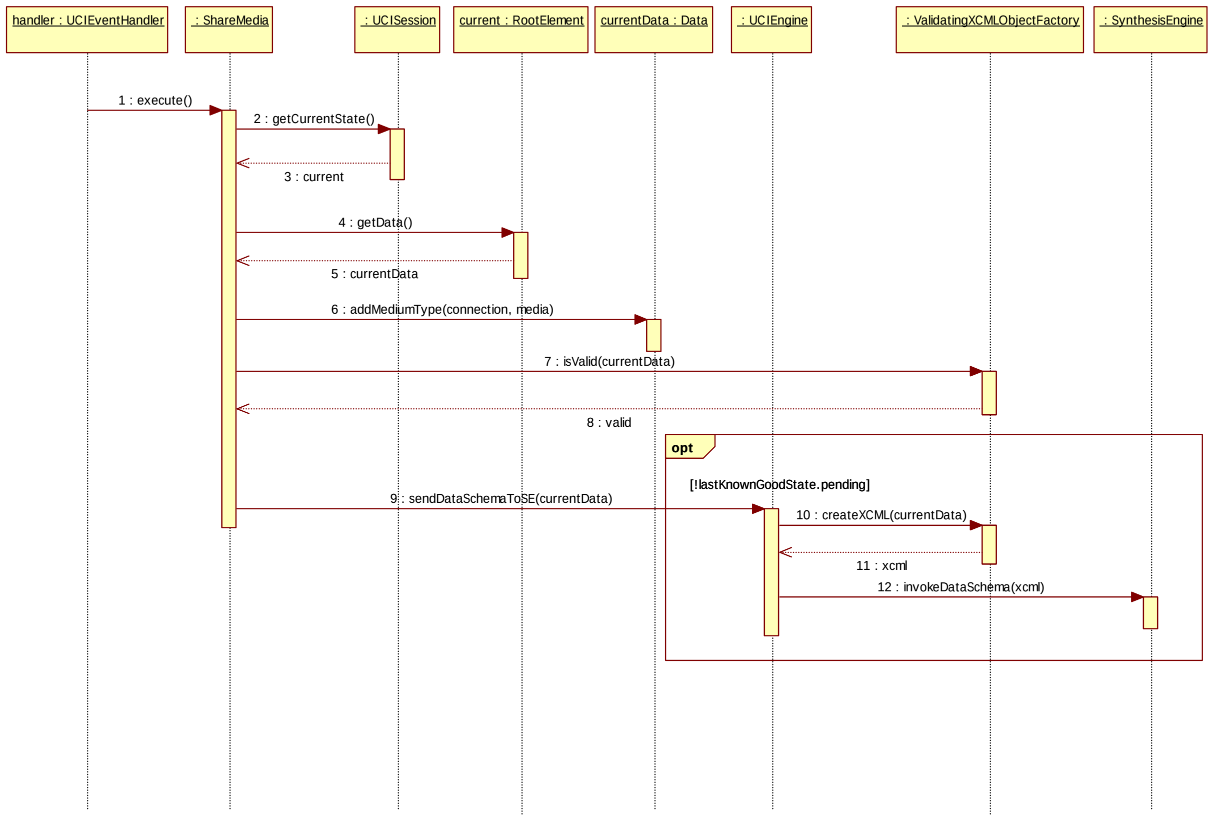
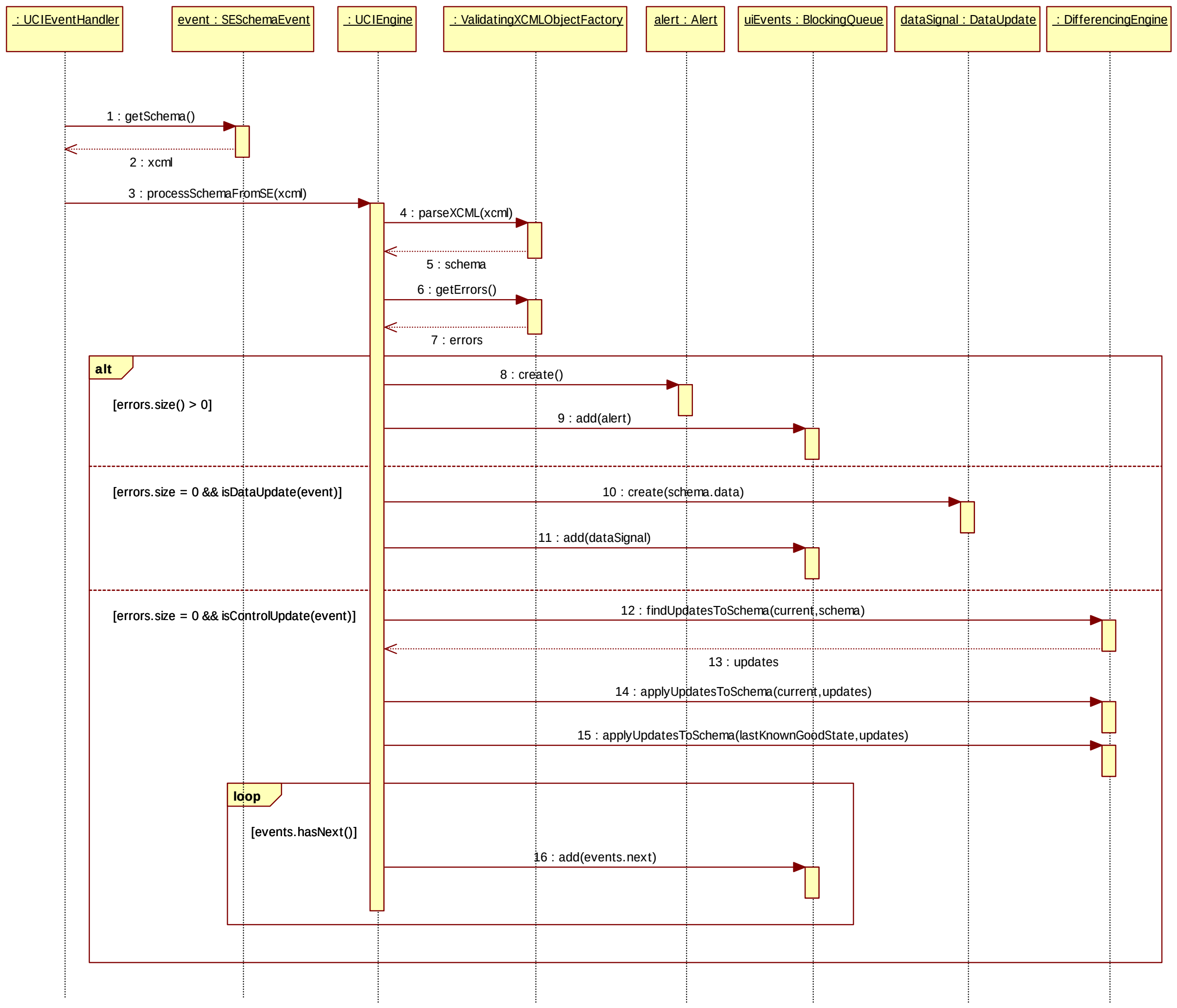

handler : UCIEventHandler
: ShareMedia
: UCISession
current : RootElement
currentData : Data
: UCIEngine
: ValidatingXCMLObjectFactory
: SynthesisEngine
1 : execute()
2 : getCurrentState()
3 : current
4 : getData()
5 : currentData
6 : addMediumType(connection, media)
7 : isValid(currentData)
8 : valid
opt
[!lastKnownGoodState.pending]
9 : sendDataSchemaToSE(currentData)
10 : createXCML(currentData)
11 : xcml
12 : invokeDataSchema(xcml)

Figure 35 Share Media Sequence

6.3.5 Synthesis Engine Updates Communication Schema

The following sequence diagram describes the sequence of interactions occurring when the Synthesis Engine sends a schema update event.



6.3.6 *JOIN COMMUNICATION*

The following sequence diagram describe the sequence of interactions occurring when the Synthesis Engine sends an event indicating that the user has been invited to join a communication.

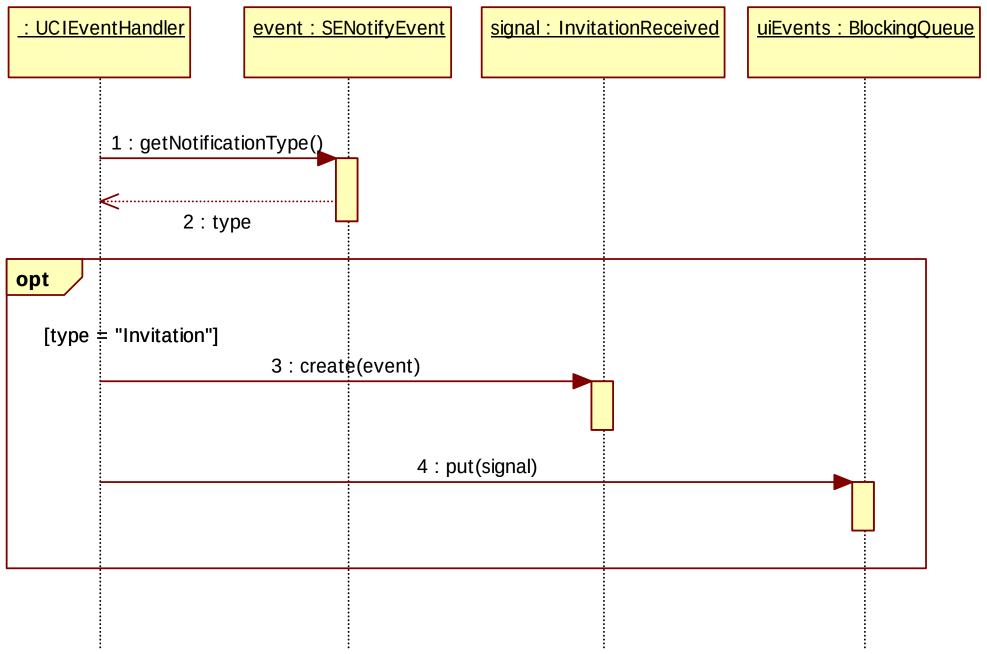


Figure 36 Join Communication Sequence

The previous diagram indicates that the UCI sends a signal to the user interface indicating that the user has been invited to join a communication. Later, the GUI controller takes that event and prompts the user to either accept or reject the invitation. The user response is then sent to the Synthesis Engine. This sequence of interactions is illustrated in the sequence diagram below.

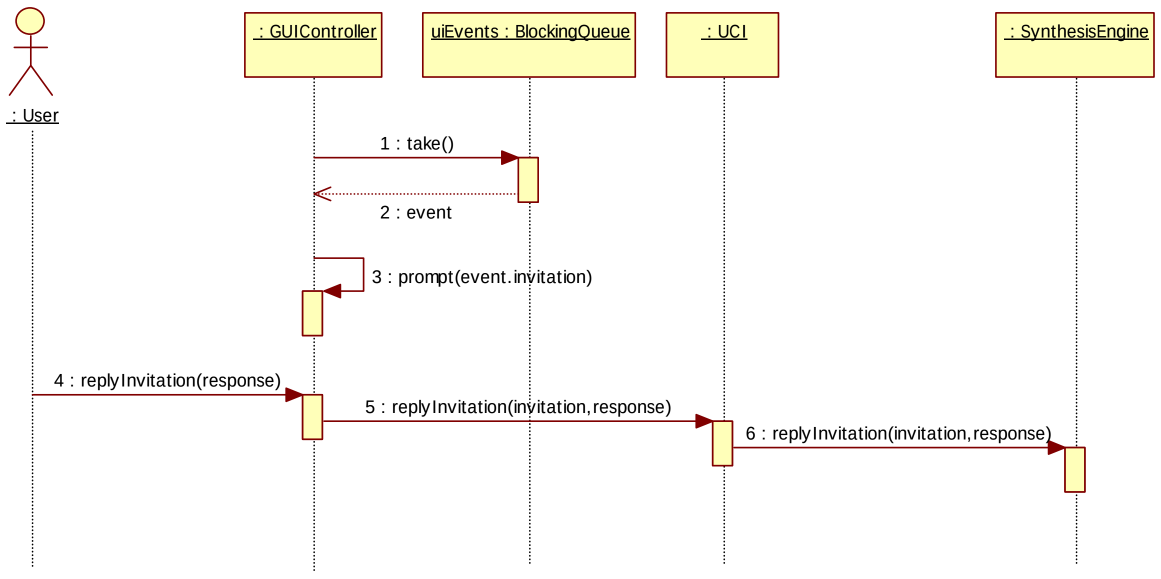

: User
: GUIController
uiEvents : BlockingQueue
: UCI
: SynthesisEngine
1 : take()
2 : event
3 : prompt(event.invitation)
4 : replyInvitation(response)
5 : replyInvitation(invitation,response)
6 : replyInvitation(invitation,response)

Figure 37 Join Communication GUI Controller Sequence

6.3.7 *Save Communication*

The following sequence diagram illustrates the object interactions for saving a communication schema on the local repository. It should be noted that this operation is synchronous, which means that there is no need to queue a command in the UCI event handler queue.

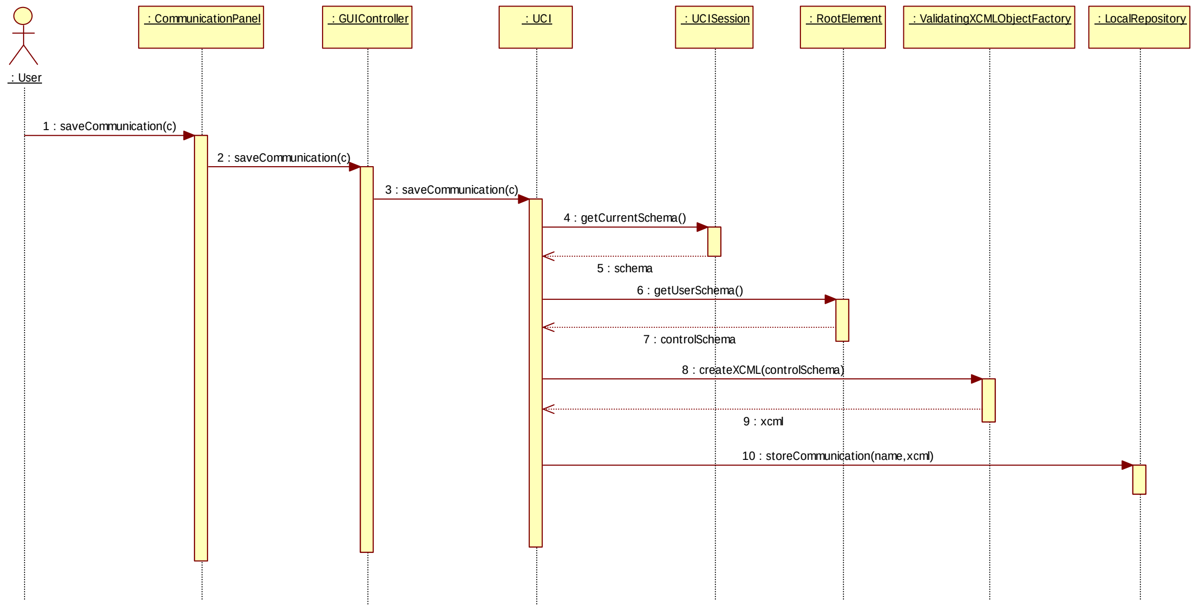


Figure 38 Save Communication Sequence

6.4 Detailed Class Design

The following subsections detail the class design in each subsystem.

6.4.1 *GUI (Appendix C Fig. 24,25,26 Appendix D p.112)*

- ChatPanel - A traditional chat control panel: history, new text, send button.
- Communication - This is the GUI class that represents the communication window. It has a list of connections, each of them represented as a tab. This class instantiates the shell and display used by all the panels.
- ConnectionTab - A tab in the communication window. It represents a connection.
- FileAndFormsPanel - A panel that shows a table with the shared files and forms.

- MediaComposite - Control that represents a participant on a connection. It shows a video monitor or picture and provides some basic controls for audio and video.

- ParticipantsPanel - Panel that shows a group of participants.

- SettingsPanel – A panel with basic controls for managing audio and video settings: audio/mic volume, audio/video enabling or disabling.

- GuiController (interface) - This is the event handling class for the GUI component; here Events get pulled up or sent down to or from the UCI. Appropriate lines of action are taken after each Event is computed.

- GuiControllerImp – This is the implementation of the GuiController interface.

- AddMediator - This class creates Add Mediator dialog box.

- EditContacts - This class creates Edit Contacts dialog box.

- EditProfile - This class creates Edit Profile dialog box.

- LogIn - This class is the log in pane for CVM.

- NewForm - This class creates New Form dialog box.

- TabPanel - This class is the main application window composed of tabs.

6.4.2 Synthesis Engine (Appendix D p. 113)

- SynthesisEngine (interface) – This is a **façade** for the Synthesis Engine. The façade exposes functions exchange XCML schemas and events with the UCI.

- SEEvent (interface) – This is the root interface for all events that originate from the Synthesis Engine.

- SENotifyEvent (interface) – This is a particular SE Event that is used to present notifications/alerts from the SE.

- SESchemaEvent (interface) – This is a particular SE Event that is used to notify the UCI of a control schema updates and new data schemas.

- SynthesisEngineImpl – This is a testing stub that mimics the required functionality presented in the SE façade.

6.4.3 UCI (Appendix C Fig. 29,30,31,32 Appendix D p. 113-115)

- UCI - This is the facade for the UCI. It exposes all the necessary methods for the SE and GUI to communicate with the UCI.
- UCIFactory – This class uses the **factory** and **singleton** patterns. This is a factory for the UCI.
- ActionType (enum) - This enum represents the possible action types in XCML.
- BuiltInType (enum) - This enum defines the supported built in types. These types are the basis for medium and media types
- UCIRequest (interface) – This is the base interface for UCI events. Implementations of this class (or of subclasses), along with UCI Controller, UCI Engine and the GUI, implement the **command pattern**.
- DataRequest (interface) - This is the base interface for UCI data requests.
- Alert (interface) - Allow the UCI to present alerts such as errors to the GUI.
- InvitationReceived (interface) - Allows UCI to prompt GUI to ask user if they wish to join a communication.
- FileShared (interface) - Allows UCI to respond to the OpenSharedFile request.
- UCISignal (interface) - This is the base interface for UCI signal events.
- AlertEnum (enum) - The types of alert that can be sent to the GUI.
- ParticipantConfirmed (interface) - Allow the UCI to notify the GUI when a newly added participant accepts an invitation.
- UCIComponent (interface) - This interface is a base interface for all UCIRequestcomponents.
- Communication (interface) – A UCIRequestcomponent that allows the GUI and UCI to create and delete a connection.
- Connection (interface) - A UCIRequestcomponent that allows the GUI to add a connection to a communication.
- Form (interface) - A UCIRequestcomponent that allows the GUI to share a form in a connection.
- FormType (interface) - A UCIRequestcomponent that allows the GUI to build new Form Types and add them to a communication.
- Media (interface) - A UCIRequestcomponent that allows the UCI/GUI to talk about instances of media.

- MediaType (interface) - A UCIRequestcomponent that allows the GUI/UCI to talk about media types.
- Medium (interface) - A UCIRequestcomponent that allows the GUI/UCI to add/remove mediums to a connection/communication.
- MediumType (interface) - A UCIRequestcomponent that allows the UCI/GUI to talk about user defined and built in medium types.
- Participant (interface) - A UCIRequestcomponent that allows the GUI/UCI to add/update/remove people to connections.
- AudioFile - This class represents an audio file.
- AudioVideoFile - This class represents an audio video file.
- BinaryFile - This class represents a binary file.
- TextFile - This class represents a Text file.
- TextMessage - This class represents a Text message.
- VideoFile - This class represents a Video file.
- LiveAudio - This class represents live audio.
- LiveAudioVideo - This class represents live audio video.
- LiveVideo - This class represents a live video.
- UCIException - Class to wrap all errors that might happen in the UCI.
- DifferencingEngine - A class that is able to extract a series of UCIEvents from two control schemas. The series of UCIEvents represent the difference between the two schema.
- UCIController – This is the class that manages the main control flow for the UCI.
- UCIEngine - This class does the work of processing the events and schemas from the upper and lower layers the UCI is attached to.
- UCIEventHandler – This class is a thread for the UCI which initiates the sending and receiving of events between the upper and lower layers the UCI is attached to.
- UCISession - This class stores all the state associated with a communication.
- ValidatingXCMLObjectFactory - This class uses the **decorator pattern**. This class add complex semantic validation by decorating an XCMLObjectFactory.

- XCMLSemanticVistor (interface) - This class uses the **visitor pattern**. This interface extends the generic XCMLVisitor for the XCML validation vistor to implement.
- XCMLSemanticValidator – This class implements the XCMLSemanticVistor interface and validates a XCML object tree.
- CommunicationImpl - This class is an implementation of a Communication.
- ConnectionImpl - This class is an implementation of a Connection.
- FormImpl - This class is an implementation of a Form.
- FormTypeImpl - This class is an implementation of a FormType.
- MediaImpl - This class is an implementation of a Media.
- MediaTypeImpl - This class is an implementation of a MediaType.
- MediumImpl - This class is an implementation of a Medium.
- MediumTypeImpl - This class is an implementation of a MediumType.
- ParticipantsImpl- This class is an implementation of a Participant.
- AddMedium – A UCI Request that allows the UCI/GUI to start/add a medium (start video/audio/etc).
- AddParticipant - A UCI Request that allows the UCI/GUI to add a participant.
- LeaveCommunication - A UCI Request that allows the GUI to leave/close a communication.
- LoadCommunication - A UCI Request that allows GUI to request the loading of a saved communication.
- RemoveMedium - A UCI Request that allows the UCI/GUI to stop/remove a medium (stop video/audio/etc).
- ReplyInvitation - A UCI Request that allows the GUI to send a invitation response to the UCI.
- ShareForm - A UCI Request that allows the GUI/UCI to share forms.
- ShareMedia - A UCI Request that allows the GUI/UCI to share media.
- StoreCommunication - A UCI Request that allows llows the GUI to request that the UCI save the communication.

6.4.4 XCML (Appendix C Fig. 27 & 28, Appendix D p. 114,115)

- ValidationError – This class represent a validation error.
- XCMLObjectFactory – This is class uses the **adapter pattern.** This class uses the generated ObjectFactory and adds syntactic and XML semantic error detection/handling with the XCMLValidationEventHandler.
- XCMLValidationEventHandler – This class handles syntactic and XML semantic errors while marshaling or un-marshaling XCML object trees.
- XCMLVisitor(interface) – This class uses the **visitor pattern** to navigate XCML object trees.
- ObjectFactory –This class uses the **factory pattern.** A class generated by JAXB based on the XSD to create XCML object trees.
- RootElement – This class wraps the UserSchema and Data node types.
- ActionType – A class generated by JAXB based on the XSD to create XCML object trees.
- CapabilityType – A class generated by JAXB based on the XSD to create XCML object trees.
- ConnectionType – A class generated by JAXB based on the XSD to create XCML object trees.
- Data – A class generated by JAXB based on the XSD to create XCML object trees.
- DeviceType – A class generated by JAXB based on the XSD to create XCML object trees.
- FormType – A class generated by JAXB based on the XSD to create XCML object trees.
- FormTypeType – A class generated by JAXB based on the XSD to create XCML object trees.
- IsAttachedType – A class generated by JAXB based on the XSD to create XCML object trees.
- MediumType – A class generated by JAXB based on the XSD to create XCML object trees.
- MediumTypeType – A class generated by JAXB based on the XSD to create XCML object trees.
- PersonType – A class generated by JAXB based on the XSD to create XCML object trees.
- StateType – A class generated by JAXB based on the XSD to create XCML object trees.
- UserSchema – A class generated by JAXB based on the XSD to create XCML object trees.

6.4.5 Repository (Appendix C Fig. 33, Appendix D p. 113)

- LocalRepository – Stores Communications, Files, and Forms on disk.

- CVMForm – A stored form type.
- CVMFile – A stored file type.

7 TESTING PROCESS

This section presents the unit, subsystem and system test suites as well as the test results for the CVM GUI.

7.1 SYSTEM TESTS

The following are the system test cases.

Test Case Id	CVMGUI_COM_001_UCI_001 - Create Communication
Purpose	The purpose of this test case is to verify that when a user runs the CVM application, it creates a communication correctly.
Pre Condition	2. The first user has logged into the system as "Peter". 3. The second user has logged into the system as "John".
Input	1.The user "Peter" creates a communication between him and the user "John".
Expected Output	•A new communication has been created containing just the user as a participant and the user's device as part of an incomplete communication. •The communication status information is being displayed to the user as well as the necessary panels for handling each device capability handled by the user's device.
Actual Output	1.A communication has been created. Communication status information is being displayed as well as all the panels for each device. PASSED

Test Case Id	CVMGUI_COM_001_UCI_002 - Create Communication
Purpose	The purpose of this test case is to verify that when a user runs the CVM application, it creates a communication correctly.
Pre Condition	1.The first user has logged into the system as "George". 2.The second user has logged into the system as "Ana".
Input	1.The user "George" creates a communication between him and the user "Ana".
Expected Output	1.A new communication has been created containing just the user as a participant and the user's device as part of an incomplete communication. 2.The communication status information is being displayed to the user as well as the necessary panels for handling each device capability handled by the user's.
Actual Output	1.A communication has been created. Communication status information is being displayed as well as all the panels for each device. PASSED

Test Case Id	CVMGUI_ COM_002_UCI_001 - Load Communication
Purpose	The purpose of this test case is to verify that when a user loads a schema, the application loads and displays the user selected schema correctly.
Pre Condition	1.The user "Peter" has logged into the system. 2.There is a saved schema, "good.xml", in the local directory of the user "Peter".

Input	1.The user “Peter” clicks on the top-left corner of the application and selects Control -> Load Communication.
	
	2.The user “Peter” browses through directory and selects “good.xml” as the communication to load. This is the content of “good.xml”:
	
	`<?xml version="1.0" encoding="UTF-8" standalone="yes"?>`
	`<userSchema communicationID="good">`
	`<connection connectionID="Connection 23387093">`
	`<device isLocal="true" deviceID="Device 19235919">`
	`<deviceCapability>AudioFile</deviceCapability>`
	`<deviceCapability>AVFile</deviceCapability>`
	`<deviceCapability>BinaryFile</deviceCapability>`
	`<deviceCapability>LiveAudio</deviceCapability>`
	`<deviceCapability>LiveAV</deviceCapability>`
	`<deviceCapability>LiveStream</deviceCapability>`
	`<deviceCapability>LiveVideo</deviceCapability>`
	`<deviceCapability>NonStreamFile</deviceCapability>`
	`<deviceCapability>StreamFile</deviceCapability>`
	`<deviceCapability>Text</deviceCapability>`
	`<deviceCapability>TextFile</deviceCapability>`
	`<deviceCapability>VideoFile</deviceCapability>`
	`</device>`
	`<device deviceID="Device 16237341">`
	`<deviceCapability>AudioFile</deviceCapability>`
	`<deviceCapability>AVFile</deviceCapability>`
	`<deviceCapability>BinaryFile</deviceCapability>`
	`<deviceCapability>LiveAudio</deviceCapability>`
	`<deviceCapability>LiveAV</deviceCapability>`
	`<deviceCapability>LiveStream</deviceCapability>`
	`<deviceCapability>LiveVideo</deviceCapability>`
	`<deviceCapability>NonStreamFile</deviceCapability>`
	`<deviceCapability>StreamFile</deviceCapability>`

	<deviceCapability>Text</deviceCapability> <deviceCapability>TextFile</deviceCapability> <deviceCapability>VideoFile</deviceCapability> </device> </connection> <person personRole="Surgeon" personID="1" personName="George"/> <person personRole="Referring Phisician" personID="2" personName="Ana"/> <isAttached deviceID="Device 19235919" personID="1"/> <isAttached deviceID="Device 16237341" personID="2"/> </userSchema>
Expected Output	The following messages are shown to standard output: Add Connection Event. Connection: Connection 23387093 Add Participant Event. Connection: Connection 23387093 Participant: George Add Participant Event. Connection: Connection 23387093 Participant: Ana
Actual Output	The following messages are shown to standard output: Add Connection Event. Connection: Connection 23387093 Add Participant Event. Connection: Connection 23387093 Participant: George Add Participant Event. Connection: Connection 23387093 Participant: Ana PASSED

Test Case Id	CVMGUI_ COM_002_UCI_002 - Load Communication
Purpose	The purpose of this test case is to verify that when a user loads a schema, the

	application loads and displays the user selected schema correctly.
Pre Condition	1.The user “Peter” has logged into the system. 2.There is a saved schema, “good.xml”, in the local directory of the user “Peter”.
Input	1.User clicks on the top-left corner of the application and selects Control -> Load Communication. 2.User browses through directory and selects “good2.xml” as the communication to load. This is the content of “good2.xml”: 3.<?xml version="1.0" encoding="UTF-8" standalone="yes"?> <userSchema communicationID="good"> <connection connectionID="Connection 23387093"> <device isLocal="true" deviceID="Device 19235919"> <deviceCapability>AudioFile</deviceCapability> <deviceCapability>AVFile</deviceCapability> <deviceCapability>BinaryFile</deviceCapability> <deviceCapability>LiveAudio</deviceCapability> <deviceCapability>LiveAV</deviceCapability> <deviceCapability>LiveStream</deviceCapability> <deviceCapability>LiveVideo</deviceCapability> <deviceCapability>NonStreamFile</deviceCapability> <deviceCapability>StreamFile</deviceCapability> <deviceCapability>Text</deviceCapability> <deviceCapability>TextFile</deviceCapability> <deviceCapability>VideoFile</deviceCapability> </device> <device deviceID="Device 16237341"> <deviceCapability>AudioFile</deviceCapability> <deviceCapability>AVFile</deviceCapability> <deviceCapability>BinaryFile</deviceCapability>

	<deviceCapability>LiveAudio</deviceCapability> <deviceCapability>LiveAV</deviceCapability> <deviceCapability>LiveStream</deviceCapability> <deviceCapability>LiveVideo</deviceCapability> <deviceCapability>NonStreamFile</deviceCapability> <deviceCapability>StreamFile</deviceCapability> <deviceCapability>Text</deviceCapability> <deviceCapability>TextFile</deviceCapability> <deviceCapability>VideoFile</deviceCapability> </device> </connection> <person personRole="User" personID="1" personName="Peter"/> <person personRole="Referring Phisician" personID="2" personName="John"/> <isAttached deviceID="Device 19235919" personID="1"/> <isAttached deviceID="Device 16237341" personID="2"/> </userSchema>
Expected Output	The following messages are printed into standard output: Add Connection Event. Connection: Connection 23387093 Add Participant Event. Connection: Connection 23387093 Participant: Peter Add Participant Event. Connection: Connection 23387093 Participant: John
Actual Output	The following messages are printed into standard output: Add Connection Event. Connection: Connection 23387093 Add Participant Event. Connection: Connection 23387093 Participant: Peter Add Participant Event. Connection: Connection 23387093 Participant: John PASSED

Test Case Id	CVMGUI_ COM_002_UCI_003 - Load Communication
Purpose	The purpose of this test case is to verify that when a user loads a schema, the application loads and displays the user selected schema correctly.
Pre Condition	1.The user "Peter" has logged into the system. 2.There is a saved schema, "corrupt.xml", in the local directory of the user "Peter".
Input	1.User clicks on the top-left corner of the application and selects Control -> Load Communication. 2.User browses through directory and selects "corrupt.xml" as the communication to load. The content of "corrupt.xml" is: `<?xml version="1.0" encoding="UTF-8" standalone="yes"?>` `<userSchema communicationID="corrupt">` `<connection connectionID="Connection 23387093">` `<device isLocal="true" deviceID="Device 19235919">` `<deviceCapability>AudioFile</deviceCapability>` `<deviceCapability>AVFile</deviceCapability>` `<deviceCapability>BinaryFile</deviceCapability>` `<deviceCapability>LiveAudio</deviceCapability>` `<deviceCapability>LiveAV</deviceCapability>` `<deviceCapability>LiveStream</deviceCapability>` `<deviceCapability>LiveVideo</deviceCapability>` `<deviceCapability>NonStreamFile</deviceCapability>` `<deviceCapability>StreamFile</deviceCapability>` `<deviceCapability>Text</deviceCapability>` `<deviceCapability>TextFile</deviceCapability>` `<deviceCapability>VideoFile</deviceCapability>` `<device deviceID="Device 16237341">`

```
            <deviceCapability>AudioFile</deviceCapability>
            <deviceCapability>AVFile</deviceCapability>
            <deviceCapability>BinaryFile</deviceCapability>
            <deviceCapability>LiveAudio</deviceCapability>
            <deviceCapability>LiveAV</deviceCapability>
            <deviceCapability>LiveStream</deviceCapability>
            <deviceCapability>LiveVideo</deviceCapability>
            <deviceCapability>NonStreamFile</deviceCapability>
            <deviceCapability>StreamFile</deviceCapability>
            <deviceCapability>Text</deviceCapability>
            <deviceCapability>TextFile</deviceCapability>
            <deviceCapability>VideoFile</deviceCapability>
        </device>
    </connection>
</userSchema>
```

Expected Output	Throws a UCI exception.
Actual Output	Throws a UCI exception. PASSED

Test Case Id	CVMGUI_ COM_005_UCI_001 - Save Communication
Purpose	The purpose of this test case is to verify that a communication can be saved in a repository.

Pre Condition	1.There is an active communication with two participants. 2.The user “Peter” is one of the participants in the connection. 3.The user “John” is one of the participants in the connection.
Input	1.The user “Peter” clicks on the top-left corner of the application and selects Control -> Save Communication. 2.The user “Peter” browses through the local directory and selects a location for the save communication to be saved and names the file “savedcommunication.xml”.
Expected Output	1.The saved communication is available on the main application window. 2.The Synthesis Engine has received the updated communication schema.
Actual Output	1.Updated schema was saved into a local repository. PASSED

Test Case Id	CVMGUI_ COM_005_UCI_002 - Save Communication
Purpose	The purpose of this test case is to verify that a communication can be saved in a repository.
Pre Condition	1.There is an active communication with 3 participants. 2.The user “Peter” is one of the participants in the connection. 3.The user “John” is one of the participants in the connection.

	4.The user “Ana” is one of the participants in the connection.
Input	1.The user “Peter” clicks on the top-left corner of the application and selects Control -> Save Communication. 2.The user “Peter” browses through the local directory and selects a location for the save communication to be saved and names the file “savedcommunication2.xml”.
Expected Output	•The saved communication is available on the main application window. •The Synthesis Engine has received the updated communication schema.
Actual Output	1.Updated schema was saved into a local repository. PASSED

Test Case Id	CVMGUI_ COM_006_UCI_001 - Add Participant
Purpose	The purpose of this test case is to verify that when a user adds a participant, the new participant is added to the communication and the schema is updated.
Pre Condition	•The user “Peter” has an active communication with at least one connection. •The user “John” is in the same communication and connection as “Peter”. •The user “William” is logged into the system. •The user “William” is in “Peter” contact list.
Input	1.The user “Peter” drags and drops “William” from the contact list to the Participant panel of the communication “Peter” is in.

Expected Output	1.The new participant has been added to the communication. 2.The synthesis engine has received the updated communication schema. 3.The new participant appears on the participant list with status pending.
Actual Output	1.User was able to add a participant to the communication and the schema updated properly and the user interface updated accordingly. PASSED

Test Case Id	CVMGUI_ COM_006_UCI_002 - Add Participant
Purpose	The purpose of this test case is to verify that when a user adds a participant, the new participant is added to the communication and the schema is updated.
Pre Condition	1.The user "Peter" has an active communication with at least one connection. 2.The user "John" is in the same communication and connection as "Peter". 3.The user "William" is in the same communication and connection as "Peter" 4.The user "Ana" is logged into the system. 5.The user "Ana" is in the user "Peter" contact list.
Input	1.The user "Peter" drags and drops "Ana" from the contact list to the Participant panel of the communication "Peter" is in.
Expected Output	1.The new participant has been added to the communication. 2.The synthesis engine has received the updated communication schema. 3.The new participant appears on the participant panel with status pending.

Actual Output	1.User was able to add a participant to the communication and the schema updated properly and the user interface updated accordingly. PASSED

Test Case Id	CVMGUI_MED_002_UCI_001 - Enable Live Audio/Video
Purpose	The purpose of this test case is to verify that when a user enables live audio/video, the schema has been updated and the user interface reflects the changes.
Pre Condition	1.The user “Peter” and “John” are in the same communication. 2.The user “Peter” has the necessary hardware for live audio/video streaming. 3.The user “John” has the necessary hardware for live audio/video streaming.
Input	1.The user “Peter” clicks the live audio/video check box on the user “John” panel.
Expected Output	1.The live audio/video medium has been enabled on the connection. 2.The updated schema has been sent to the synthesis engine for negotiation. 3.The live audio/video panel indicates that live audio/video is enabled on the selected connection.
Actual Output	1.User was able to enable live video/audio in the connection. PASSED

Test Case Id	CVMGUI_MED_002_UCI_002 – Enable Live Audio/Video
Purpose	The purpose of this test case is to verify that when a user enables live audio/video, the schema has been updated and the user interface reflects the changes.
Pre Condition	1.The user “Peter” and “Ana” are in the same communication. 2.The user “Peter” has the necessary hardware for live audio/video streaming. 3.The user “Ana” has the necessary hardware for live audio/video streaming.
Input	1.The user “Peter” clicks the live audio/video check box on the user “Ana” panel.
Expected Output	1.The live audio/video medium has been enabled on the connection. 2.The updated schema has been sent to the synthesis engine for negotiation. 3.The live audio/video panel indicates that live audio/video is enabled on the selected connection.
Actual Output	1.User was able to enable live video/audio in the connection. PASSED

Test Case Id	CVMGUI_MED_002_UCI_003 – Enable Live Audio/Video
Purpose	The purpose of this test case is to verify that when a user enables live audio/video on an already user with live audio/video streaming, the schema has been updated and the user interface reflects the changes.

Pre Condition	1.The user “Peter” and “Ana” are in the same communication. 2.The user “Peter” has the necessary hardware for live audio/video streaming. 3.The user “Ana” has the necessary hardware for live audio/video streaming. 4.The live audio/video medium is already enabled on the user “Ana” connection selected for enabling
Input	1.The user “Peter” clicks the live audio/video check box on the user “Ana” panel.
Expected Output	1.The live audio/video medium has been enabled on the connection. 2.The updated schema has been sent to the synthesis engine for negotiation. 3.The live audio/video panel indicates that live audio/video is enabled on the selected connection.
Actual Output	1.User was able to enable live video/audio in the connection. PASSED

Test Case Id	CVMGUI_MED_003_UCI_001 - Disable Live Audio/Video
Purpose	The purpose of this test case is to verify that when a user disables live audio/video, the schema has been updated and the user interface reflects the changes.
Pre Condition	1.The user “Peter” has started a communication and there is at least one connection with a minimum of two active participants. 2.The user “John” is in the same communication as “Peter”. 3.Both user “Peter” and “John” have the necessary hardware for live audio/video

	streaming. 4.The live audio/video medium is enabled on the connection selected for disabling.
Input	1.The user “Peter” clicks the live audio/video check box on the user “John” panel
Expected Output	1.The live audio/video medium has been removed from the user “John” connection. 2.The updated schema has been sent to the synthesis engine for negotiation. 3.The live audio/video panel indicates that live audio/video is disabled on the selected connection.
Actual Output	1.User was able to stop live video/audio in the connection. PASSED

Test Case Id	CVMGUI_MED_003_UCI_002 - Disable Live Audio/Video
Purpose	The purpose of this test case is to verify that when a user disables live audio/video, the schema has been updated and the user interface reflects the changes.
Pre Condition	1.The user “Peter” has started a communication and there is at least one connection with a minimum of two active participants. 2.The user “John” is in the same communication as “Peter”. 3.The user “Ana” is in the same communication as “Peter”.

	4.Both user “Peter”, “John”, and “Ana” have the necessary hardware for live audio/video streaming. 5.The live audio/video medium is enabled on the connection selected for disabling.
Input	1.The user “Peter” clicks the live audio/video check box on the user “Ana” panel
Expected Output	1.The live audio/video medium has been removed from the user “Ana” connection. 2.The updated schema has been sent to the synthesis engine for negotiation. 3.The live audio/video panel indicates that live audio/video is disabled on the selected connection.
Actual Output	1.User was able to stop live video/audio in the connection. PASSED

Test Case Id	CVMGUI_MED_003_UCI_003 - Disable Live Audio/Video
Purpose	The purpose of this test case is to verify that when a user disable live audio/video on a user who already has disabled live audio/video streaming, the schema has been updated and the user interface reflects the changes.
Pre Condition	1.The user “Peter” has started a communication and there is at least one connection with a minimum of two active participants. 2.The user “John” is in the same communication as “Peter”. 3.Both user “Peter” and “John” have the necessary hardware for live audio/video

	streaming. 4.The live audio/video medium is already disabled on the user “John” connection selected for disabling.
Input	•The user “Peter” clicks the live audio/video check box on the user “John” panel
Expected Output	1.The live audio/video medium has been removed from the user “John” connection. 2.The updated schema has been sent to the synthesis engine for negotiation. 3.The live audio/video panel indicates that live audio/video is disabled on the selected connection.
Actual Output	5.User was able to stop live video/audio in the connection. PASSED

Test Case Id	CVMGUI_MED_004_UCI_001 - Share Media
Purpose	The purpose of this test case is to verify that when a user selects a media to be shared, all participants can view the shared media.
Pre Condition	1.The user “Peter” has started a communication and there is a connection already established between two or more participants. 2.The user “John” is in the same communication that “Peter” is in. 3.The user “Peter” has a file called “project.doc” in his File Cabinet tab.
Input	1.The user “Peter” drags and drops the file “project.doc” from his File Cabinet tab

	into the Files panel in the communication window he is with user "John".
Expected Output	1.The communication schema has been updated so the selected media is now being shared. 2.The Synthesis Engine has received and negotiated the updated schema. 3.The system has updated the communication schema with the schema received from the Synthesis Engine. 4.The user interface reflects the negotiated schema. 5.The selected media is now being shared by all the participants in the connection.
Actual Output	1.User was able to add a media to the share panel and the user interface reflected the changes of the schema and all participants in the connection were able to see the share media. PASSED

Test Case Id	CVMGUI_MED_004_UCI_002 - Share Media
Purpose	The purpose of this test case is to verify that when a user selects a media to be shared, all participants can view the shared media.
Pre Condition	1.The user "Peter" has started a communication and there is a connection already established between two or more participants. 2.The user "John" is in the same communication that "Peter" is in. 3.The user "Ana" is in the same communication that "Peter" is in.

	4.The user “Peter” has a file called “project.doc” in his File Cabinet tab.
Input	•The user “Peter” drags and drops the file “project.doc” from his File Cabinet tab into the Files panel in the communication window he is with user “John” and “Ana”.
Expected Output	1.The communication schema has been updated so the selected media is now being shared. 2.The Synthesis Engine has received and negotiated the updated schema. 3.The system has updated the communication schema with the schema received from the Synthesis Engine. 4.The user interface reflects the negotiated schema. 5.The selected media is now being shared by all the participants in the connection.
Actual Output	1.User was able to add a media to the share panel and the user interface reflected the changes of the schema and all participants in the connection were able to see the share media. PASSED

Test Case Id	CVMGUI_MED_005_UCI_001 - Start Live Audio/Video
Purpose	The purpose of this test case is to verify that when a user starts live audio/video, the schema has been updated and the user interface reflects the changes.
Pre Condition	1.The user “Peter” has started a communication and there is at least one connection with a minimum of two active participants.

	2.The user "John" is in the same communication as "Peter".
Input	1.The user "Peter" clicks on the button to start live audio/video streaming.
Expected Output	1.The live audio/video medium has been added to the connection. 2.The updated schema has been sent to the synthesis engine for negotiation. 3.The live audio/video panel indicates that live audio/video is enabled on the selected connection.
Actual Output	1.User was able to start live audio/video in the connection.PASSED

Test Case Id	CVMGUI_MED_005_UCI_002 - Stop Live Audio/Video
Purpose	The purpose of this test case is to verify that when a user stop live audio/video, the schema has been updated and the user interface reflects the changes.
Pre Condition	1.The user "Peter" has started a communication and there is at least one connection with a minimum of two active participants. 2.The user "John" is in the same communication as "Peter".
Input	2.The user "Peter" clicks on the button to stop live audio/video streaming.
Expected Output	4.The live audio/video medium has been added to the connection. 5.The updated schema has been sent to the synthesis engine for negotiation. 6.The live audio/video panel indicates that live audio/video is enabled on the selected connection.

Actual Output	2.User was able to start live audio/video in the connection.PASSED

Test Case Id	CVMGUI_MED_005_UCI_003 - Start Live Audio/Video
Purpose	The purpose of this test case is to verify that when a user starts live audio/video when live audio/video already is enabled, the schema has been updated and the user interface reflects the changes.
Pre Condition	1.The user "Peter" has started a communication and there is at least one connection with a minimum of two active participants. 2.The user "John" is in the same communication as "Peter". 3.Live audio/video has already been started by user "Peter".
Input	3.The user "Peter" clicks on the button to start live audio/video streaming.
Expected Output	7.The live audio/video medium has been added to the connection. 8.The updated schema has been sent to the synthesis engine for negotiation. 9.The live audio/video panel indicates that live audio/video is enabled on the selected connection.
Actual Output	3.User was able to start live audio/video in the connection.PASSED

Test Case Id	CVMGUI_MED_006_UCI_001 - Send Chat Message
Purpose	The purpose of this test case is to verify that when a user sends a chat message,

	the other participants receives it.
Pre Condition	1.The user “Peter” has started a communication and there is at least one connection with a minimum of two active participants. 2.The user “John” is in the same communication as “Peter”.
Input	1.The user “Peter” enters “hello world” into the Chat panel and clicks the Send button.
Expected Output	1.The communication schema has been updated so the chat message gets sent in the selected connection. 2.The Synthesis Engine has received and negotiated the updated schema. 3.The system has updated the communication schema with the schema received from the Synthesis Engine. 4.The chat message sent appears on the chat panel.
Actual Output	1.User was able to send chat messages to all participants in the connection. PASSED

Test Case Id	CVMGUI_MED_006_UCI_002 - Send Chat Message
Purpose	The purpose of this test case is to verify that when a user sends a chat message, the other participants receives it.
Pre Condition	1.The user “Peter” has started a communication and there is at least one connection with a minimum of two active participants.

	2.The user "John" is in the same communication as "Peter". 3.The user "Ana" is in the same communication as "Peter".
Input	1.The user "Peter" enters "hello world 2" into the Chat panel and clicks the Send button.
Expected Output	1.The communication schema has been updated so the chat message gets sent in the selected connection. 2.The Synthesis Engine has received and negotiated the updated schema. 3.The system has updated the communication schema with the schema received from the Synthesis Engine. 4.The chat message sent appears on the chat panel.
Actual Output	1.User was able to send chat messages to all participants in the connection. PASSED

7.2 Subsystem Tests

The following are the subsystem test cases.

Test Case Id	CVMGUI_COM_001_UCI_001 - Create Communication
Purpose	The purpose of this test case is to verify that a communication gets properly created.
Pre Condition	There is a test driver for the UCI subsystem. There is a stub for the Synthesis Engine. There is participant instance with connection id = 1, username = "Peter", role

	="Faculty" There is a second participant instance with connection id = 2, username = "John", role = "Student"
Input	The test driver invokes the createUCI() method of the UCIFactory class with the following arguments and returns a new UCI instance: •A new empty queue. •The id of the local participant: 1. The test driver invokes the createCommunication() method of the uci created from the UCIFactory class with no argument. The test driver invokes the createConnection() method of the uci passing in the previously created communication and returns a Connection. The test driver invokes the addParticipant() passing in the first participant and the previously created connection instance. The test driver invokes the addParticipant() passing in the first participant and the previously created connection instance.

Expected Output	The standard output contains the following communication schema:
	
	`<userSchema communicationID="Communication 1">`
	`    <connection bandwidth="" connectionID="Connection 6166383">`
	`        <device isLocal="true" isVirtual="false"`
	`deviceID="Device 13480046">`
	`            <deviceCapability>AudioFile</deviceCapability>`
	`            <deviceCapability>AVFile</deviceCapability>`
	`            <deviceCapability>BinaryFile</deviceCapability>`
	`            <deviceCapability>LiveAudio</deviceCapability>`
	`            <deviceCapability>LiveAV</deviceCapability>`
	`            <deviceCapability>LiveStream</deviceCapability>`
	`            <deviceCapability>LiveVideo</deviceCapability>`
	`            <deviceCapability>NonStreamFile</deviceCapability>`
	`            <deviceCapability>StreamFile</deviceCapability>`
	`            <deviceCapability>Text</deviceCapability>`
	`            <deviceCapability>TextFile</deviceCapability>`
	`            <deviceCapability>VideoFile</deviceCapability>`
	`        </device>`
	`        <device isLocal="false" isVirtual="false"`
	`deviceID="Device 31335791">`
	`            <deviceCapability>AudioFile</deviceCapability>`
	`            <deviceCapability>AVFile</deviceCapability>`
	`            <deviceCapability>BinaryFile</deviceCapability>`
	`            <deviceCapability>LiveAudio</deviceCapability>`

```
            <deviceCapability>LiveAV</deviceCapability>
            <deviceCapability>LiveStream</deviceCapability>
            <deviceCapability>LiveVideo</deviceCapability>
            <deviceCapability>NonStreamFile</deviceCapability>
            <deviceCapability>StreamFile</deviceCapability>
            <deviceCapability>Text</deviceCapability>
            <deviceCapability>TextFile</deviceCapability>
            <deviceCapability>VideoFile</deviceCapability>
        </device>
    </connection>
    <person personRole="User" personID="1" personName="Peter"/>
    <person personRole="Student" personID="2"
personName="John"/>
    <isAttached deviceID="Device 13480046" personID="1"/>
    <isAttached deviceID="Device 31335791" personID="2"/>
</userSchema>
```

Actual Output

```
<userSchema communicationID="Communication 1">

    <connection bandwidth="" connectionID="Connection 6166383">
        <device isLocal="true" isVirtual="false"
deviceID="Device 13480046">
            <deviceCapability>AudioFile</deviceCapability>
            <deviceCapability>AVFile</deviceCapability>
            <deviceCapability>BinaryFile</deviceCapability>
            <deviceCapability>LiveAudio</deviceCapability>
            <deviceCapability>LiveAV</deviceCapability>
            <deviceCapability>LiveStream</deviceCapability>
            <deviceCapability>LiveVideo</deviceCapability>
            <deviceCapability>NonStreamFile</deviceCapability>
            <deviceCapability>StreamFile</deviceCapability>
            <deviceCapability>Text</deviceCapability>
            <deviceCapability>TextFile</deviceCapability>
            <deviceCapability>VideoFile</deviceCapability>
        </device>
        <device isLocal="false" isVirtual="false"
deviceID="Device 31335791">
            <deviceCapability>AudioFile</deviceCapability>
            <deviceCapability>AVFile</deviceCapability>
            <deviceCapability>BinaryFile</deviceCapability>
            <deviceCapability>LiveAudio</deviceCapability>
            <deviceCapability>LiveAV</deviceCapability>
            <deviceCapability>LiveStream</deviceCapability>
            <deviceCapability>LiveVideo</deviceCapability>
            <deviceCapability>NonStreamFile</deviceCapability>
            <deviceCapability>StreamFile</deviceCapability>
            <deviceCapability>Text</deviceCapability>
            <deviceCapability>TextFile</deviceCapability>
            <deviceCapability>VideoFile</deviceCapability>
        </device>
    </connection>
    <person personRole="User" personID="1" personName="Peter"/>
    <person personRole="Student" personID="2"
personName="John"/>
```

	`<isAttached deviceID="Device 13480046" personID="1"/>`
	`<isAttached deviceID="Device 31335791" personID="2"/>`
	`</userSchema>`
	
	PASS

Test Case Id	CVMGUI_COM_001_UCI_002 - Create Communication
Purpose	The purpose of this test case is to verify that a communication gets properly created.
Pre Condition	There is a test driver for the UCI subsystem. There is a stub for the Synthesis Engine. There are 2 participant instances with the following data There is participant instance with connection id = 1, username = “George”, role =”Surgeon” There is a second participant instance with connection id = 2, username = “Ana”, role = “Referring Physician”
Input	The test driver invokes the createUCI() method of the UCIFactory class with the following arguments and returns a new UCI instance: •A new empty queue. •The id of the local participant: 1. The test driver invokes the createCommunication() method of the uci created from the UCIFactory class with no argument. The test driver invokes the createConnection() method of the uci passing in the previously created communication and returns a Connection. The test driver invokes the addParticipant() passing in the first participant and the previously created connection instance. The test driver invokes the addParticipant() passing in the first participant and

	the previously created connection instance.

Expected Output

Standard output contains the following communication schema:

```
<userSchema communicationID="Communication 1">
    <connection bandwidth="" connectionID="Connection 6166383">
        <device isLocal="true" isVirtual="false"
deviceID="Device 13480046">
            <deviceCapability>AudioFile</deviceCapability>
            <deviceCapability>AVFile</deviceCapability>
            <deviceCapability>BinaryFile</deviceCapability>
            <deviceCapability>LiveAudio</deviceCapability>
            <deviceCapability>LiveAV</deviceCapability>
            <deviceCapability>LiveStream</deviceCapability>
            <deviceCapability>LiveVideo</deviceCapability>
            <deviceCapability>NonStreamFile</deviceCapability>
            <deviceCapability>StreamFile</deviceCapability>
            <deviceCapability>Text</deviceCapability>
            <deviceCapability>TextFile</deviceCapability>
            <deviceCapability>VideoFile</deviceCapability>
        </device>
        <device isLocal="false" isVirtual="false"
deviceID="Device 31335791">
            <deviceCapability>AudioFile</deviceCapability>
            <deviceCapability>AVFile</deviceCapability>
            <deviceCapability>BinaryFile</deviceCapability>
            <deviceCapability>LiveAudio</deviceCapability>
            <deviceCapability>LiveAV</deviceCapability>
            <deviceCapability>LiveStream</deviceCapability>
            <deviceCapability>LiveVideo</deviceCapability>
            <deviceCapability>NonStreamFile</deviceCapability>
            <deviceCapability>StreamFile</deviceCapability>
            <deviceCapability>Text</deviceCapability>
            <deviceCapability>TextFile</deviceCapability>
            <deviceCapability>VideoFile</deviceCapability>
        </device>
    </connection>
    <person personRole="Surgeon" personID="1"
personName="George"/>
    <person personRole="Referring Physician" personID="2"
personName="Ana"/>
    <isAttached deviceID="Device 13480046" personID="1"/>
    <isAttached deviceID="Device 31335791" personID="2"/>
</userSchema>
```

Actual Output

```
<userSchema communicationID="Communication 1">
    <connection bandwidth="" connectionID="Connection 6166383">
        <device isLocal="true" isVirtual="false"
deviceID="Device 13480046">
            <deviceCapability>AudioFile</deviceCapability>
            <deviceCapability>AVFile</deviceCapability>
            <deviceCapability>BinaryFile</deviceCapability>
            <deviceCapability>LiveAudio</deviceCapability>
            <deviceCapability>LiveAV</deviceCapability>
            <deviceCapability>LiveStream</deviceCapability>
            <deviceCapability>LiveVideo</deviceCapability>
            <deviceCapability>NonStreamFile</deviceCapability>
            <deviceCapability>StreamFile</deviceCapability>
            <deviceCapability>Text</deviceCapability>
            <deviceCapability>TextFile</deviceCapability>
            <deviceCapability>VideoFile</deviceCapability>
        </device>
        <device isLocal="false" isVirtual="false"
deviceID="Device 31335791">
            <deviceCapability>AudioFile</deviceCapability>
            <deviceCapability>AVFile</deviceCapability>
            <deviceCapability>BinaryFile</deviceCapability>
            <deviceCapability>LiveAudio</deviceCapability>
            <deviceCapability>LiveAV</deviceCapability>
            <deviceCapability>LiveStream</deviceCapability>
            <deviceCapability>LiveVideo</deviceCapability>
            <deviceCapability>NonStreamFile</deviceCapability>
            <deviceCapability>StreamFile</deviceCapability>
            <deviceCapability>Text</deviceCapability>
            <deviceCapability>TextFile</deviceCapability>
            <deviceCapability>VideoFile</deviceCapability>
        </device>
    </connection>
    <person personRole="Surgeon" personID="1"
personName="George"/>
    <person personRole="Referring Physician" personID="2"
personName="Ana"/>
    <isAttached deviceID="Device 13480046" personID="1"/>
    <isAttached deviceID="Device 31335791" personID="2"/>
</userSchema>
```

```
PASS
```

Test Case Id	CVMGUI_ COM_002_SUB_001 - Load Communication
Purpose	The purpose of this test case is to verify that when a user loads a schema the events get properly generated.
Pre Condition	There is a test driver for the UCI. There is a stub for the Synthesis Engine.

The good.xcml file contains the following schema:

```
<userSchema communicationID="Communication 1">
    <connection bandwidth="" connectionID="Connection 6166383">
        <device isLocal="true" isVirtual="false"
deviceID="Device 13480046">
            <deviceCapability>AudioFile</deviceCapability>
            <deviceCapability>AVFile</deviceCapability>
            <deviceCapability>BinaryFile</deviceCapability>
            <deviceCapability>LiveAudio</deviceCapability>
            <deviceCapability>LiveAV</deviceCapability>
            <deviceCapability>LiveStream</deviceCapability>
            <deviceCapability>LiveVideo</deviceCapability>
            <deviceCapability>NonStreamFile</deviceCapability>
            <deviceCapability>StreamFile</deviceCapability>
            <deviceCapability>Text</deviceCapability>
            <deviceCapability>TextFile</deviceCapability>
            <deviceCapability>VideoFile</deviceCapability>
        </device>
        <device isLocal="false" isVirtual="false"
deviceID="Device 31335791">
            <deviceCapability>AudioFile</deviceCapability>
            <deviceCapability>AVFile</deviceCapability>
            <deviceCapability>BinaryFile</deviceCapability>
            <deviceCapability>LiveAudio</deviceCapability>
            <deviceCapability>LiveAV</deviceCapability>
            <deviceCapability>LiveStream</deviceCapability>
            <deviceCapability>LiveVideo</deviceCapability>
            <deviceCapability>NonStreamFile</deviceCapability>
            <deviceCapability>StreamFile</deviceCapability>
            <deviceCapability>Text</deviceCapability>
            <deviceCapability>TextFile</deviceCapability>
            <deviceCapability>VideoFile</deviceCapability>
        </device>
    </connection>
    <person personRole="Surgeon" personID="1"
personName="George"/>
    <person personRole="Referring Physician" personID="2"
personName="Ana"/>
    <isAttached deviceID="Device 13480046" personID="1"/>
    <isAttached deviceID="Device 31335791" personID="2"/>
</userSchema>
```

Input	The test driver invokes the createUCI() method of the UCIFactory class with the following arguments and returns a new UCI called “uci”: The queue is the empty queue event from the stub. The test driver invokes the loadCommunication() method of the uci with the following arguments passing in the “good.xml” file.

Expected Output	The standard output contains the XCML for the “good.xcml” schema.
Actual Output	The standard output contains the XCML for the “good.xcml” schema.

Test Case Id	CVMGUI_ COM_005_SUB_001 - Save Communication
Purpose	The purpose of this test case is to verify that a communication can be saved in a repository.
Pre Condition	There is a test driver for the Login subsystem. There is a stub with 2 participant classes with the following data: The first participant connection id = 1, username = “Peter”, role =”Faculty” The second participant connection id = 2, username = “John”, role = “Student”. There is a stub for the event queues that has been configured with empty events.
Input	The test driver invokes the createUCI() method of the UCIFactory class with the following arguments and returns a new UCI called “uci”: The queue is the empty queue event from the stub. The test driver invokes the createCommunication() method of the uci created from the UCIFactory class with no argument. The test driver invokes the createConnection() method of the uci with the following arguments and returns a Connection called “connection”: The communication created in step 2. The test driver invokes the addParticipant() method twice of the uci with the following arguments: The first participant connection id = 1, username = “Peter”, role =”Faculty”

The second participant connection id = 2, username = “John”, role = “Student”

The connection created in step 3 and the first and second participant from the stub.

The test driver invokes the storeCommunication() method of the uci with the following argument:

The communication created in step 2.

Expected Output

The schema is saved by the repository: there is a file, “C://tmp/Communication 1”, with the following content.

```
<userSchema communicationID="Communication 1">
    <connection bandwidth="" connectionID="Connection 6166383">
        <device isLocal="true" isVirtual="false"
deviceID="Device 13480046">
            <deviceCapability>AudioFile</deviceCapability>
            <deviceCapability>AVFile</deviceCapability>
            <deviceCapability>BinaryFile</deviceCapability>
            <deviceCapability>LiveAudio</deviceCapability>
            <deviceCapability>LiveAV</deviceCapability>
            <deviceCapability>LiveStream</deviceCapability>
            <deviceCapability>LiveVideo</deviceCapability>
            <deviceCapability>NonStreamFile</deviceCapability>
            <deviceCapability>StreamFile</deviceCapability>
            <deviceCapability>Text</deviceCapability>
            <deviceCapability>TextFile</deviceCapability>
            <deviceCapability>VideoFile</deviceCapability>
        </device>
        <device isLocal="false" isVirtual="false"
deviceID="Device 31335791">
            <deviceCapability>AudioFile</deviceCapability>
            <deviceCapability>AVFile</deviceCapability>
            <deviceCapability>BinaryFile</deviceCapability>
            <deviceCapability>LiveAudio</deviceCapability>
            <deviceCapability>LiveAV</deviceCapability>
            <deviceCapability>LiveStream</deviceCapability>
            <deviceCapability>LiveVideo</deviceCapability>
            <deviceCapability>NonStreamFile</deviceCapability>
            <deviceCapability>StreamFile</deviceCapability>
            <deviceCapability>Text</deviceCapability>
            <deviceCapability>TextFile</deviceCapability>
            <deviceCapability>VideoFile</deviceCapability>
        </device>
    </connection>
    <person personRole="Surgeon" personID="1"
personName="Peter"/>
    <person personRole="Referring Physician" personID="2"
personName="John"/>
    <isAttached deviceID="Device 13480046" personID="1"/>
    <isAttached deviceID="Device 31335791" personID="2"/>
```

	`</userSchema>`
	
	`PASS`

Actual Output	The specified file was saved with the correct XCML content. PASSED

Test Case Id	CVMGUI_MED_005_SUB_001 - Enable Live Audio/Video
Purpose	The purpose of this test case is to verify that when a user enable live audio/video, the schema has been updated and the user interface reflects the changes.
Pre Condition	There is a test driver for the Login subsystem. There is a stub for the event queues that has been configured with empty events. There is a stub with 2 participant classes with the following data : The first participant connection id = 1, username = “Peter”, role = “Faculty”. The first participant connection id = 1, username = “John”, role = “Student”. There is a stub with a MediumType class type with the following data: The type is LIVE_AUDIO_VIDEO. There is a stub with Medium class type with the following data: The type is the MediumType from step #4.
Input	The test driver invokes the createUCI() method of the UCIFactory class with the following arguments and returns a new UCI called “uci”: The queue is the empty queue event from the stub The test driver invokes the createCommunication() method of the uci created from the UCIFactory class with no argument. The test driver invokes the createConnection() method of the uci with the

	following arguments and returns a Connection called “connection”: The communication created in step 2. The test driver invokes the addParticipant() method three times of the uci with the following arguments: The connection created in step 3 and the first, second and third participant from the stub. The test driver invokes the addMedium() method of the uci with the following arguments: The connection from step #3. The medium from the stub.
Expected Output	The live audio/video medium has been added to the connection. The updated schema has been sent to the synthesis engine for negotiation. The following schema is printed into standard output:
Actual Output	User was able to start live audio/video in the connection. PASSED

Test Case Id	CVMGUI_MED_003_SUB_001 - Disable Live Audio/Video
Purpose	The purpose of this test case is to verify that when a user disables live audio/video, the schema has been updated and the user interface reflects the changes.
Pre Condition	There is a test driver for the Login subsystem. There is a stub for the event queues that has been configured with empty events.

	There is a stub with 2 participant classes with the following data : The first participant connection id = 1, username = “Peter”, role = “Faculty”. The first participant connection id = 1, username = “John”, role = “Student”. There is a stub with a MediumType class type with the following data: The type is LIVE_AUDIO_VIDEO. There is a stub with Medium class type with the following data: The type is the MediumType from step #4.
Input	The test driver invokes the createUCI() method of the UCIFactory class with the following arguments and returns a new UCI called “uci”: The queue is the empty queue event from the stub The test driver invokes the createCommunication() method of the uci created from the UCIFactory class with no argument. The test driver invokes the createConnection() method of the uci with the following arguments and returns a Connection called “connection”: The communication created in step 2. The test driver invokes the addParticipant() method three times of the uci with the following arguments: The connection created in step 3 and the first, second and third participant from the stub. The test driver invokes the addMedium() method of the uci with the following arguments: The connection from step #3. The medium from the stub. The test driver invokes the removeMedium() method of the uci with the following arguments: The connection from step #3. The medium from the stub.

Expected Output	The live audio/video medium has been removed from the connection. The updated schema has been sent to the synthesis engine for negotiation. The live audio/video panel indicates that live audio/video is disabled on the selected connection.
Actual Output	User was able to stop live video/audio in the connection. PASSED

Test Case Id	CVMGUI_MED_001_SUB_001 - Share Media
Purpose	The purpose of this test case is to verify that when a user selects a media to be shared, all participants can view the shared media.
Pre Condition	There is a test driver for the Login subsystem. There is a stub for the event queues that has been configured with empty events. There is a stub with 2 participant classes with the following data : The first participant connection id = 1, username = “Peter”, role = “Faculty”. The first participant connection id = 1, username = “John”, role = “Student”. There is a stub with a Media class type with the following data: The action type is ActionType SEND. The text file is new TextFile(). The build in type is is BuiltInType.TEXT_FILE.value().
Input	The test driver invokes the createUCI() method of the UCIFactory class with the following arguments and returns a new UCI called “uci”: The queue is the empty queue event from the stub The test driver invokes the createCommunication() method of the uci created

	from the UCIFactory class with no argument. The test driver invokes the createConnection() method of the uci with the following arguments and returns a Connection called “connection”: The communication created in step 2. The test driver invokes the addParticipant() method three times of the uci with the following arguments: The connection created in step #3 and the first, second and third participant from the stub. The test driver invokes the shareMedia() method of the uci with the following arguments: The connection created in step #3. The media from the stub.
Expected Output	The communication schema has been updated so the selected media is now being shared. The Synthesis Engine has received and negotiated the updated schema. The system has updated the communication schema with the schema received from the Synthesis Engine. The user interface reflects the negotiated schema. The selected media is now being shared by all the participants in the connection.
Actual Output	User was able to add a media to the share panel and the user interface reflected the changes of the schema and all participants in the connection were able to see the share media. PASSED

Test Case Id	CVMGUI_MED_006_SUB_001 - Send Chat Message
Purpose	The purpose of this test case is to verify that when a user sends a chat message,

	the other participants receives it.
Pre Condition	There is a test driver for the Login subsystem. There is a stub for the event queues that has been configured with empty events. There is a stub with 2 participant classes with the following data : The first participant connection id = 1, username = “Peter”, role = “Faculty”. The first participant connection id = 1, username = “John”, role = “Student”. There is a stub with a Media class type with the following data: The action type is ActionType.SEND. The text file is new TextMessage(). The build in type is is BuiltInType.TEXT_MSG.value().
Input	The test driver invokes the createUCI() method of the UCIFactory class with the following arguments and returns a new UCI called “uci”: The queue is the empty queue event from the stub The test driver invokes the createCommunication() method of the uci created from the UCIFactory class with no argument. The test driver invokes the createConnection() method of the uci with the following arguments and returns a Connection called “connection”: The communication created in step 2. The test driver invokes the addParticipant() method three times of the uci with the following arguments: The connection created in step #3 and the first, second and third participant from the stub. The test driver invokes the shareMedia() method of the uci with the following arguments: The connection created in step #3. The media from the stub.

Expected Output	The communication schema has been updated so the chat message gets sent in the selected connection. The Synthesis Engine has received and negotiated the updated schema. The system has updated the communication schema with the schema received from the Synthesis Engine. The chat message sent appears on the chat panel.
Actual Output	User was able to send chat messages to all participants in the connection. PASSED

7.3 UNIT TESTS

The state-based unit tests were performed on the UCI controller to verify that the each transition is correctly performed. A test driver and stubs were created to minimize the coupling between the UCI controller and the rest of the system.

Test Case Id	CVMGUI_UCI_UTC_001 – No Communication
Purpose	The purpose of this test case is to verify that after creating a new UCIController object, the state of the UCIController is in the “No Communication” state.
Pre Condition	1 A test driver has been created to call the UCIController constructor. 2 Test stubs have been created for UCIStrategyFactory, LocalRepository, XCMLObjectFactory, UCIEventHandler, and UCIException.
Input	1. The test driver calls the UCIController constructor passing it a BlockingQueue and the String “userID”.
Expected Output	2.The UCIController should be in the “No Communication” state.

Actual Output	•The UCIController is in the "No Communication" state. PASSED

Test Case Id	CVMGUI_UCI_UTC_002 –Communication Created
Purpose	The purpose of this test case is to verify that after calling the createCommunication method, the state of the UCIController is in the "Communication Created" state.
Pre Condition	– A test driver has been created to call the createCommunication method. – Test stubs have been created for RootElement, UserSchema, XCMLObjectFactory, UCIEventHandler, UCIEngine, and CommunicationImpl. – The state of the UCIController is in the "No Communication" or the "Communication Closed" state.
Input	1. The test driver calls the createCommunication method.
Expected Output	1. The UCIController should be in the "No Communication" state.
Actual Output	• The UCIController is in the "Communication Created" state. PASSED

Test Case Id	CVMGUI_UCI_UTC_003 – Communication Updated
Purpose	The purpose of this test case is to verify that after calling the createConnection method, the state of the UCIController is in the "Communication Updated" state.

Pre Condition	1. A test driver has been created to call the createConnection method. 2. Test stubs have been created for Connection, ConnectionImpl, Comunication, UCIEventHandler, RootElement, UserSchema, and ParticipantImpl. 3. The state of the UCIController is in the “Communication Created” or the “Communication Updated” state.
Input	• The test driver calls the createConnection method.
Expected Output	1. The UCIController should be in the “Communication Updated” state.
Actual Output	– The UCIController is in the “Communication Updated” state. PASSED

Test Case Id	CVMGUI_UCI_UTC_004 – Communication Updated
Purpose	The purpose of this test case is to verify that after calling the removeMedium method, the state of the UCIController is in the “Communication Updated” state.
Pre Condition	1. A test driver has been created to call the removeMedium method. 2. Test stubs have been created for Connection, ConnectionImpl, Medium, MediumImpl, RemoveMediumEvent, UCIEvent, and UCIEventHandler. 3. The state of the UCIController is in the “Communication Created” or the “Communication Updated” state.
Input	1. The test driver calls the removeMedium method passing it a new ConnectionImpl and a new MediumStub object.
Expected	1. The UCIController should be in the “Communication Updated” state.

Output	
Actual Output	– The UCIController is in the "Communication Updated" state. PASSED

Test Case Id	CVMGUI_UCI_UTC_005 – Communication Updated
Purpose	The purpose of this test case is to verify that after calling the shareMedia method, the state of the UCIController is in the "Communication Updated" state.
Pre Condition	1. A test driver has been created to call the shareMedia method. 2. Test stubs have been created for Connection, ConnectionImpl, Media, MediaImpl, UCIException, UCIEventHandler, ShareMediaEvent, and UCIEvent 3. The state of the UCIController is in the "Communication Created" or the "Communication Updated" state.
Input	1. The test driver calls the shareMedia method passing it a new ConnectionImpl and a new MediaImpl object.
Expected Output	1. The UCIController should be in the "Communication Updated" state.
Actual Output	1. The UCIController is in the "Communication Updated" state. PASSED

Test Case Id	CVMGUI_UCI_UTC_006 – Communication Updated

Purpose	The purpose of this test case is to verify that after calling the startMedia method, the state of the UCIController is in the "Communication Updated" state.
Pre Condition	1. A test driver has been created to call the startMedia method. 2. Test stubs have been created for Connection, ConnectionImpl, Media, MediaImpl, UCIEventHandler, StartMediaEvent, and UCIEvent. 3. The state of the UCIController is in the "Communication Created" or the "Communication Updated" state.
Input	1. The test driver calls the startMedia method passing it a new ConnectionImpl and a new MediaImpl.
Expected Output	1. The UCIController should be in the "Communication Updated" state.
Actual Output	1. The UCIController is in the "Communication Updated" state. PASSED

Test Case Id	CVMGUI_UCI_UTC_007 – Communication Updated
Purpose	The purpose of this test case is to verify that after calling the stopMedia method, the state of the UCIController is in the "Communication Updated" state.
Pre Condition	• A test driver has been created to call the stopMedia method. • Test stubs have been created for Connection, ConnectionImpl, Media, MediaImpl, UCIEventHandler, StartMediaEvent, and UCIEvent. • The state of the UCIController is in the "Communication Created" or the "Communication Updated" state.

Input	– The test driver calls the stopMedia method passing it a new ConnectionImpl and a new MediaImpl.
Expected Output	1. The UCIController should be in the “Communication Updated” state.
Actual Output	1. The UCIController is in the “Communication Updated” state. PASSED

Test Case Id	CVMGUI_UCI_UTC_008 – Communication Closed
Purpose	The purpose of this test case is to verify that after calling the leaveCommunication method, the state of the UCIController is in the “Communication Closed” state.
Pre Condition	1. A test driver has been created to call the createConnection method. 2. Test stubs have been created for Connection, ConnectionImpl, Comunication, UCIEventHandler, RootElement, UserSchema, and ParticipantImpl. 3. The state of the UCIController is in the “Communication Created” or the “Communication Updated” state.
Input	1. The test driver calls the leaveCommunication method passing it a new CommunicationImpl object.
Expected Output	1. The UCIController should be in the “Communication Closed” state.
Actual Output	1. The UCIController is in the “Communication Closed” state. PASSED

7.4 Evaluation of tests

This sub-section presents the results of testing the CVM GUI.

Test Case ID	Pass	Fail
CVMGUI_COM_001_SUB_001	X	
CVMGUI_COM_002_SUB_001	X	
CVMGUI_COM_005_SUB_001	X	
CVMGUI_COM_006_SUB_001	X	
CVMGUI_MED_005_SUB_001	X	
CVMGUI_MED_003_SUB_001	X	
CVMGUI_MED_001_SUB_001	X	
CVMGUI_MED_006_SUB_001	X	

Test Case ID	Pass	Fail
CVMGUI_UCI_UTC_001	X	
CVMGUI_UCI_UTC_002	X	
CVMGUI_UCI_UTC_003	X	
CVMGUI_UCI_UTC_004	X	
CVMGUI_UCI_UTC_005	X	
CVMGUI_UCI_UTC_006	X	
CVMGUI_UCI_UTC_007	X	
CVMGUI_UCI_UTC_008	X	

Test Case ID	Pass	Fail
CVMGUI_COM_001_UCI_001	X	
CVMGUI_COM_001_UCI_002	X	
CVMGUI_COM_002_UCI_001	X	
CVMGUI_COM_002_UCI_002	X	
CVMGUI_COM_002_UCI_003	X	
CVMGUI_COM_005_UCI_001		X
CVMGUI_COM_005_UCI_002		X
CVMGUI_COM_006_UCI_001	X	
CVMGUI_COM_006_UCI_002	X	
CVMGUI_COM_006_UCI_003	X	
CVMGUI_MED_002_UCI_001	X	
CVMGUI_MED_002_UCI_002	X	
CVMGUI_MED_002_UCI_003	X	
CVMGUI_MED_003_UCI_001	X	
CVMGUI_MED_003_UCI_002	X	
CVMGUI_MED_004_UCI_001	X	
CVMGUI_MED_004_UCI_002	X	
CVMGUI_MED_005_UCI_001	X	
CVMGUI_MED_005_UCI_002	X	
CVMGUI_MED_005_UCI_003	X	
CVMGUI_MED_006_UCI_001	X	
CVMGUI_MED_006_UCI_002	X	

8 GLOSSARY

actor: External entity that interacts with the system.

analysis model: A model of the system that aims to be correct, complete, consistent, and unambiguous. The analysis model consists of the functional model, the analysis object model, and the dynamic model.

API: Application Programming Interface.

Application Programming Interface: Set of fully specified operation provided by a subsystem.

architectural pattern: Expresses a fundamental structural organization schema for software systems.

capability: The functionality provided by the device from the user is connected to the CVM (e.g. video streaming, audio streaming).

central repository: Used to store non-user specific data for the CVM.

class diagram: UML notation representing the structure of the system in terms of objects, classes, attributes, operations, and associations. Class diagrams are used to represent object models during development.

CML: Communication Modeling Language

Communication Modeling Language: Designed to meet the requirements of simplicity, network independence, and expressiveness, the CML is used to define a communication schema as well as a communication instance.

communication: Represents a set of connections between users of the CVM.

components: A physical and replaceable part of the system that complies to an interface.

connection: Represents a set of media transfers between users of the CVM.

contact: Another user which the current user can establish a connection with.

control flow: The sequence of execution of operations in the system.

control object: An object that represents a task performed by the user and supported by the system.

coupling: The strength of the dependencies between two subsystems or two classes.

criticality: Rating assigned to a piece of functionality that represents the level of failure in the event of an error.

CVM: Communication Virtual Machine.

data storage: Memory, components, devices and media that retain digital computer data used for computing for some interval of time.

DD: Design Document.

dependency: The degree to which each program module relies on each one of the other modules, represented by coupling.

deployment model: Represents run-time components and their assignments to hardware nodes.

deployment: All of the activities that make a software system available for use.

design document: A comprehensive software design model consisting of four distinct but interrelated activities: data design, architectural design, interface design, and procedural design.

design pattern: Provides a schema for refining the subsystems or components of a software system, or relations between them.

device: A computer from where a user logs into the CVM.

Eclipse: A software platform comprising extensible application frameworks, tools and a runtime library for software development and management. Written primarily in Java to provide software developers and administrators an integrated development environment (IDE).

end user: A role of the people who will use the delivered system.

extensible markup language: - A general purpose specification for creating custom markup languages.

form: An abstraction of a set of files or data, forms can be of two types: generic and specific.

frequency: Rating assigned to the piece of functionality that represents the level of invocations a particular piece of functionality will be subjected to.

functional requirements: An area of functionality the system must have.

generic form: A set of files grouped together by a user.

GUI: Graphical User Interface.

implementation model: Resulting product from the translation of the object model into code.

interface: An abstraction that an entity provides of itself to the outside. Separates the methods of external communication from internal operation, and allows it to be internally modified without affecting the way outside entities interact with it, as well as procide multiple abstrations of itself.

JAXB: XML Bindings for JAVA. See https://jaxb.dev.java.net/.

layered architecture: Software architecture that uses a logical structuring mechanism to separate the elements that make up the software solution.

local repository: Area where user's personal data will be stored, typically the user's hard disk.

media: Any video, audio, text or combination thereof.

medium: A data piece or data stream.

model: An abstraction of a system aimed at simplifying the reasoning about the system by omitting irrelevant details.

NCB: Network Communication Broker.

non-functional requirements: a constraint on the system

participant: User involved in some connection of a communication

pattern: Addresses a recurring design problem that arises in specific design situations, and presents a solution to it.

performance: Any quantifiable attribute of the system.

persistent data: Data that outlives a single execution of the system.

post condition: A condition or predicate that must always be true just after the exection of some section of code or after an operation in a formal specification.

precondition: A condition or predicate that must always be true just prior to the exectiuon of some section of code or before an operation in a formal specification

reliability: Ability of a system or component to perform its required functions under state conditions for a specified period of time.

repository architecture: An architectural pattern where subsystems access and modify a single data structure called the central repository structure.

risk: An area of uncertainty that can lead to a deviation in the project plan, including failure of the project.

schema: A description of a type of XML document, typically expressed in terms of constraints on the structure and content of documents of that type.

SE: Synthesis Engine.

sequence diagram: UML notation representing the behavior of the system as a series of interactions among a group of objects.

software requirements document:

specific form: Data retrieved from an external data source.

specification: Describes the requested behavior of the software system.

Standard Widget Toolkit: A graphical widget toolkit for use with the Java platform.

state machine: Represents the behavior of an individual object as a number of states and transitions between these states.

subsystem decomposition: The division of the system into subsystems. Each subsystem is described in terms of its services during system design and its API during object design.

subsystem: A smaller, simpler part of a larger system; in system design, a well-defined software component that provides a number of services to other subsystems.

SWT: Standard Widget Toolkit.

Synthesis Engine: External system responsible for handling the interaction between the CVM and the auxiliary communication programs. It provides a suite of algorithms to automatically synthesize a user communication schema instance to an executable form called communication control script.

UCI: User Communication Interface.

UCM: User-Centric Communication Middleware.

usability: Ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component.

USDP: Unified Software Development Process.

use case model: Represents the functionality of the system in terms of a sequence of interactions between an actor and the system.

use case: General sequence of events that describe all possible actions between actor and the system for a given piece of functionality

User Communication Interface: Provides an enviroment for end users to specify the communication schema instance to an executable form called communication control script.

user: End user or application.

X-CML schema: An XML document that contains the information of a communication. X-CML is the declarative language used for sharing information between the UCI and the Synthesis Engine.

X-CML: XML based CML.

XML: Extensible Markup Language

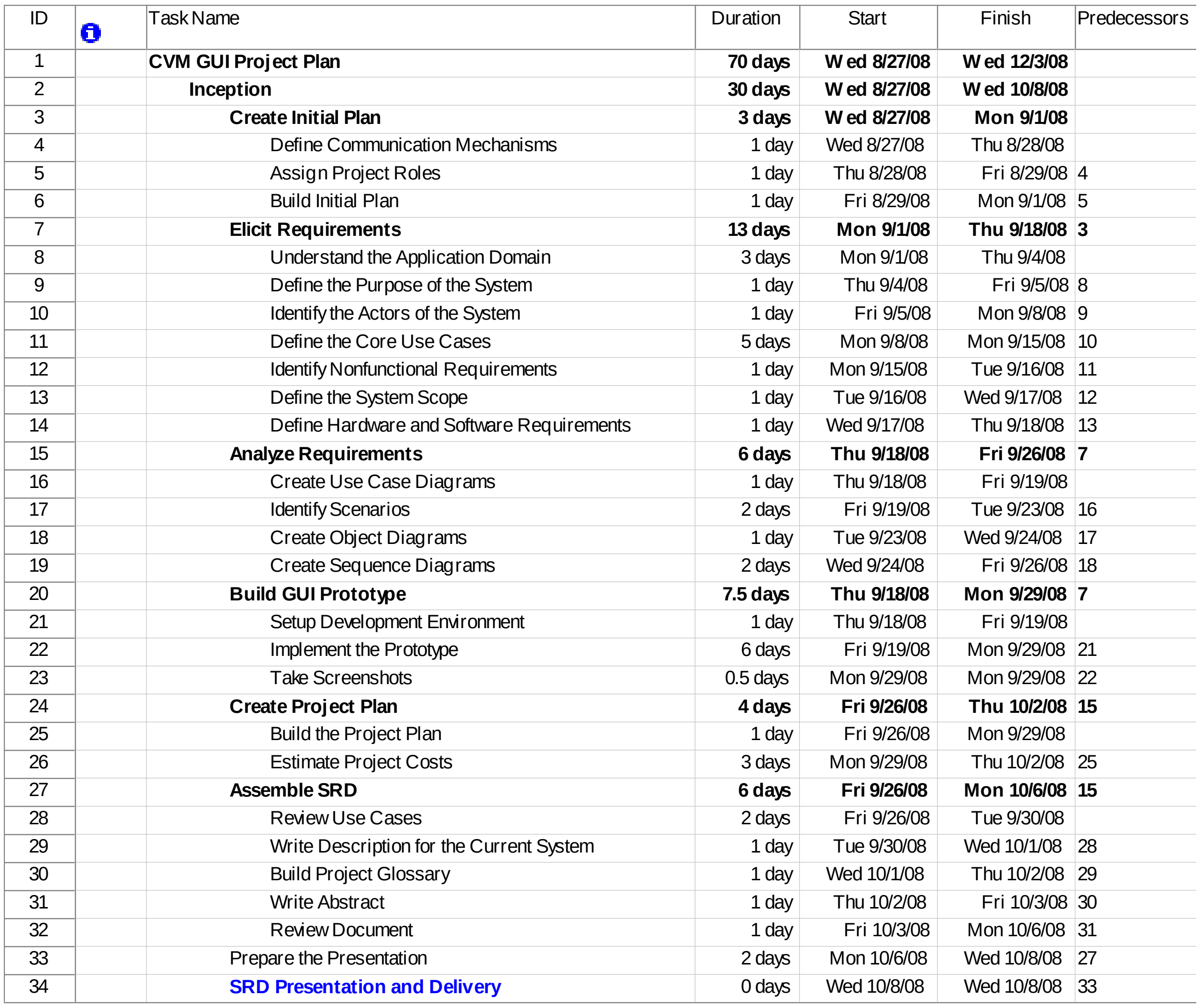
9 APPROVALS

Name	Date	Signature
Barbara Espinoza		
Jorge Guerra		
Eddie Incer		
Ricardo Koller		
David Martinez		
Hong Soong		
Nathanael Van Vorst		

10 Appendix

10.1 Appendix A – Project Schedule

This appendix presents the project schedule and Gantt charts.



ID	ℹ	Task Name	Duration	Start	Finish	Predecessors
1		**CVM GUI Project Plan**	**70 days**	**W ed 8/27/08**	**W ed 12/3/08**	
2		**Inception**	**30 days**	**W ed 8/27/08**	**W ed 10/8/08**	
3		**Create Initial Plan**	**3 days**	**W ed 8/27/08**	**Mon 9/1/08**	
4		Define Communication Mechanisms	1 day	Wed 8/27/08	Thu 8/28/08	
5		Assign Project Roles	1 day	Thu 8/28/08	Fri 8/29/08	4
6		Build Initial Plan	1 day	Fri 8/29/08	Mon 9/1/08	5
7		**Elicit Requirements**	**13 days**	**Mon 9/1/08**	**Thu 9/18/08**	**3**
8		Understand the Application Domain	3 days	Mon 9/1/08	Thu 9/4/08	
9		Define the Purpose of the System	1 day	Thu 9/4/08	Fri 9/5/08	8
10		Identify the Actors of the System	1 day	Fri 9/5/08	Mon 9/8/08	9
11		Define the Core Use Cases	5 days	Mon 9/8/08	Mon 9/15/08	10
12		Identify Nonfunctional Requirements	1 day	Mon 9/15/08	Tue 9/16/08	11
13		Define the System Scope	1 day	Tue 9/16/08	Wed 9/17/08	12
14		Define Hardware and Software Requirements	1 day	Wed 9/17/08	Thu 9/18/08	13
15		**Analyze Requirements**	**6 days**	**Thu 9/18/08**	**Fri 9/26/08**	**7**
16		Create Use Case Diagrams	1 day	Thu 9/18/08	Fri 9/19/08	
17		Identify Scenarios	2 days	Fri 9/19/08	Tue 9/23/08	16
18		Create Object Diagrams	1 day	Tue 9/23/08	Wed 9/24/08	17
19		Create Sequence Diagrams	2 days	Wed 9/24/08	Fri 9/26/08	18
20		**Build GUI Prototype**	**7.5 days**	**Thu 9/18/08**	**Mon 9/29/08**	**7**
21		Setup Development Environment	1 day	Thu 9/18/08	Fri 9/19/08	
22		Implement the Prototype	6 days	Fri 9/19/08	Mon 9/29/08	21
23		Take Screenshots	0.5 days	Mon 9/29/08	Mon 9/29/08	22
24		**Create Project Plan**	**4 days**	**Fri 9/26/08**	**Thu 10/2/08**	**15**
25		Build the Project Plan	1 day	Fri 9/26/08	Mon 9/29/08	
26		Estimate Project Costs	3 days	Mon 9/29/08	Thu 10/2/08	25
27		**Assemble SRD**	**6 days**	**Fri 9/26/08**	**Mon 10/6/08**	**15**
28		Review Use Cases	2 days	Fri 9/26/08	Tue 9/30/08	
29		Write Description for the Current System	1 day	Tue 9/30/08	Wed 10/1/08	28
30		Build Project Glossary	1 day	Wed 10/1/08	Thu 10/2/08	29
31		Write Abstract	1 day	Thu 10/2/08	Fri 10/3/08	30
32		Review Document	1 day	Fri 10/3/08	Mon 10/6/08	31
33		Prepare the Presentation	2 days	Mon 10/6/08	Wed 10/8/08	27
34		**SRD Presentation and Delivery**	0 days	Wed 10/8/08	Wed 10/8/08	33

Figure 39 Project Schedule for the Inception Phase

ID	ℹ	Task Name	Duration	Start	Finish	Predecessors
35		**Elaboration**	**20 days**	**Wed 10/8/08**	**Wed 11/5/08**	**2**
36		Build Requirements List	1 day	Wed 10/8/08	Thu 10/9/08	
37		Build the Architectural Model	5 days	Thu 10/9/08	Thu 10/16/08	36
38		Build the Deployment Model	1 day	Thu 10/16/08	Fri 10/17/08	37
39		Build the Detailed Object Model	7 days	Fri 10/17/08	Tue 10/28/08	38
40		Implement Interfaces	1 day	Tue 10/28/08	Wed 10/29/08	39
41		Assemble the SDD	4 days	Wed 10/29/08	Tue 11/4/08	40
42		Update Construction Plan	1 day	Tue 11/4/08	Wed 11/5/08	41
43		**SDD Delivery**	0 days	Wed 11/5/08	Wed 11/5/08	42
44		**Construction**	**20 days**	**Wed 11/5/08**	**Wed 12/3/08**	**35**
45		Implement the System	12 days	Wed 11/5/08	Fri 11/21/08	
46		Test the System	2 days	Fri 11/21/08	Tue 11/25/08	45
47		Assemble the Final Document	4 days	Tue 11/25/08	Mon 12/1/08	46
48		Create the User's Guide	3 days	Tue 11/25/08	Fri 11/28/08	46
49		Prepare the Final Presentation	2 days	Mon 12/1/08	Wed 12/3/08	48,47
50		**Final Deliverable and Presentation**	0 days	Wed 12/3/08	Wed 12/3/08	49

Figure 40 Project Schedule for the Elaboration and Construction Phases

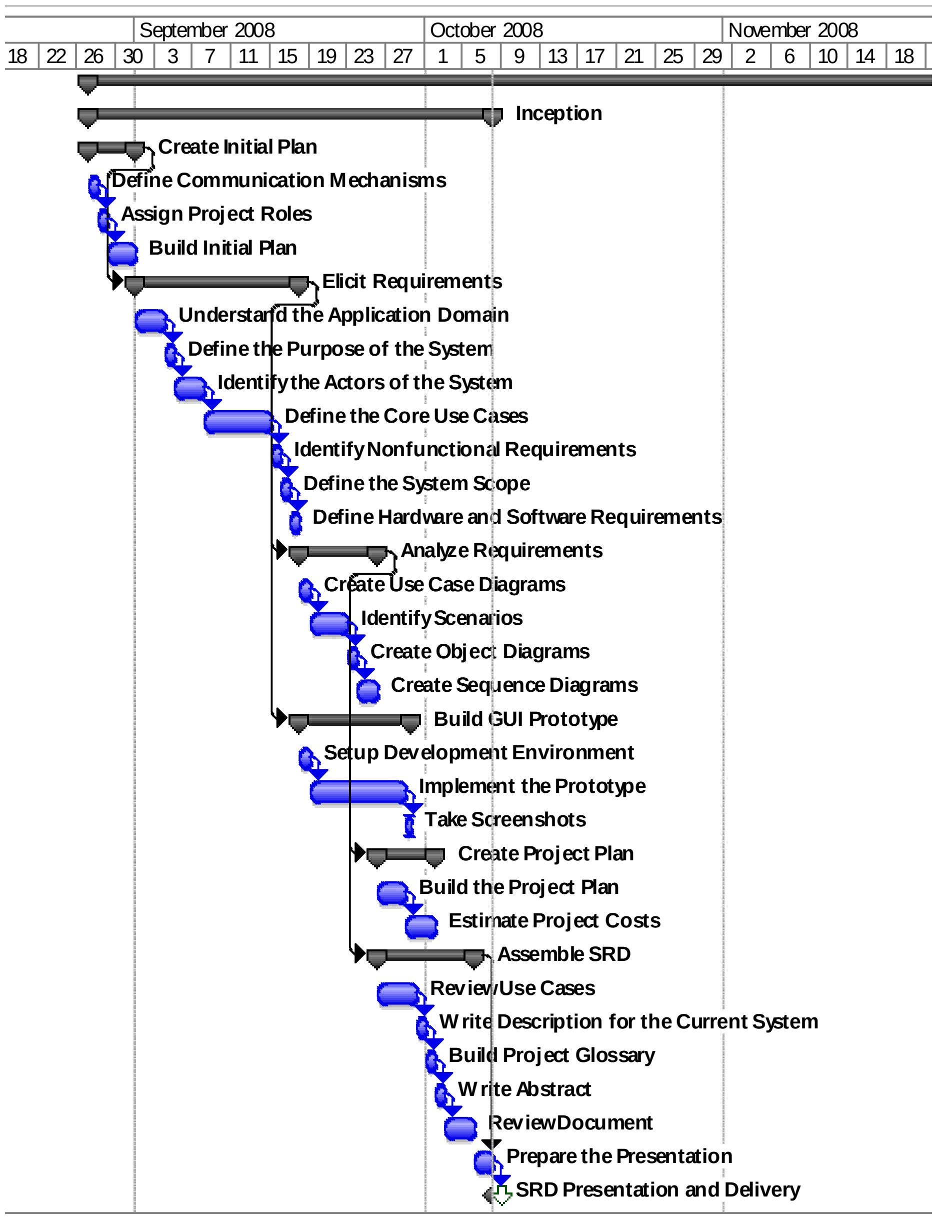


Figure 41 Gantt chart for the Inception Phase

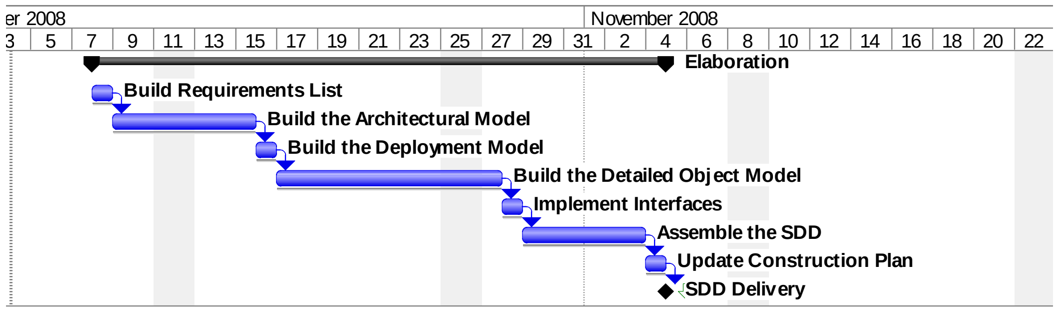


Figure 42 Gantt chart for the elaboration phase

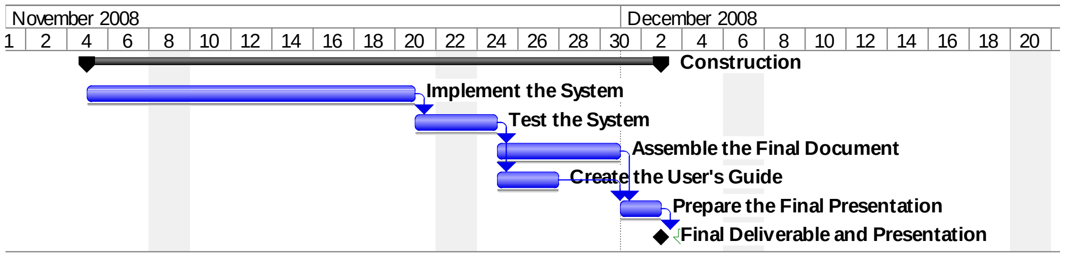


Figure 43 Gantt chart for the Construction Phase

10.2 Appendix B – Use Cases

The following subsections present the textual specification of the use cases for the CVM GUI.

10.2.1 CVMGUI_COM_001 – Create Communication

Details:

Actor: User

Pre-conditions:

1.The user has logged into the system.

Description:

1.Use case begins when the user chooses to create a new communication (See Appendix C Screenshot 21).

2.The system creates a new empty communication.

3.The system adds the user as a participant to the communication.

4.The system adds the user's devices and their capabilities to the communication.

5.The system displays the communication status information indicating that it has not yet been started and that the user is a participant in a communication.

6.The system displays the necessary panels for handling each of the available capabilities.

7.The system provides the user with a control for starting the communication.

Post-conditions:

2.A new communication has been created containing just the user as a participant and the user's device as part of an incomplete communication.

3.The communication status information is being displayed to the user as well as the necessary panels for handling each device capability handled by the user's device.

4.The system has provided the user with a control for starting the communication.

Related Use Cases:

4.Start Communication

5.Add Participant

Decision Support:

Frequency: a user is expected to create a communication 5 times per application execution.

Criticality: high.

Risk: medium.

Constraints:

Usability:

•A user without experience shall be able to create a new communication in less than 2 minutes.

Performance:

•The system shall finish creating a new communication in less than 100 milliseconds.

Modification History:

Owner: Barbara Espinoza

Initiation Date: 09/28/2008

Date Last Modified: 09/28/2008

10.2.2 CVMGUI_ COM_002 - LOAD COMMUNICATION

Details:

Actors: User, Synthesis Engine.

Pre-conditions:

4.The user has logged into the system.

5.There is at least one stored communication schema for the user.

Description:

5.Use case begins when the user chooses a stored communication schema to be loaded (See Appendix C Screenshot 18).

6.The system loads the selected communication schema.

7.The system validates the communication schema.

8.The system submits the X-CML schema to the Synthesis Engine.

9.The Synthesis Engine performs the necessary negotiations to notify all participants of the communication.

10.The system displays the list of participants of each connection.

11.The system displays the panels needed for using all the available device capabilities on each connection.

Post-conditions:

2.The communication schema is loaded and displayed to the user.

3.The Synthesis Engine has received the X-CML schema of the communication.

Exceptions:

10.If on step 3 the communication schema is invalid, the user gets notified.

Related Use Cases:

2.Validate Communication Schema.

Decision Support:

Frequency: a user is expected to load a communication schema 5 times per application execution

Criticality: high.

Risk: high.

Constraints:

Usability:

- A user without experience shall be able to load a communication in less than 2 minutes.

Reliability:

- The system shall gracefully recover 100% of the times a communication schema is invalid.

Performance:

- The system shall finish loading a communication schema in less than 1 second.

Modification History:

Owner: Barbara Espinoza

Initiation Date: 09/14/2008

Date Last Modified: 09/28/2008

10.2.3 CVMGUI_ COM_003 - JOIN CONNECTION

Details:

Actor: User, Synthesis Engine.

Pre-conditions:

1. The user has been invited by another user to join a connection.

Description:

1. Use case begins when the Synthesis Engine indicates the system that the user has been invited to join a connection.
2. The system presents the user with an invitation to join the connection (See Appendix C Screenshot 20).
3. The user chooses to accept the invitation.
4. The system informs the Synthesis Engine that the user has accepted the invitation.

5.The Synthesis Engine submits the negotiated communication schema to the system.

6.The system validates the communication schema.

7.The system displays the list of participants of the communication.

8.The system displays the panels needed for using all the available device capabilities.

Post-conditions:

1.The user is added as a participant to the communication.

2.The negotiated communication schema is loaded and displayed to the user.

Alternative Courses of Action:

1.If on step 2 the user rejects the invitation, the system discards the invitation and informs the Synthesis Engine of the user's decision.

Exceptions:

1.If on step 6 the communication schema is invalid, the user gets notified.

Related Use Cases:

1.Load Communication

2.Add Participant

Decision Support:

Frequency: a user is expected to join a communication schema 5 times per application execution.

Criticality: high.

Risk: high.

Constraints:

Usability:

- A user without experience shall be able to join a communication in less than 15 seconds.

Performance:

- The Synthesis Engine should be informed of the user's decision no later 100 milliseconds after the user indicates it.
- The system should display the negotiated communication in less than 5 seconds after it is received from the Synthesis Engine.

Modification History:

Owner: Barbara Espinoza

Initiation Date: 09/14/2008

Date Last Modified: 10/06/2008

10.2.4 CVMGUI_ COM_004 - Leave Communication

Details:

Actor: User, Synthesis Engine

Pre-conditions:

1.There is an active communication.

Description:

3.Use case begins when the user chooses the close the communication window (See Appendix C Screenshot 19).

4.The system closes the communication window, and updates the X-CML schema which is handled to the Synthesis engine.

Post-conditions:

1.The communication window is not displayed.

2.The Synthesis Engine has received the updated communication schema.

Alternative Courses of Action:

1. If in step 2, the previous state of the communication is different than the current the application will display Save Communication dialog prompting the user to "Save", "Discard", or "Cancel". Selecting "Save" invoke the use case CVMGUI_COM_005. Selecting "Discard",

the system will close the communication window and update the X-CML. And, selecting "Cancel" will return control to the user without any additional action.

Related Uses Case:

1. Save Communication.

Decision Support:

Frequency: On average a user will close 25 communications daily.

Criticality: High

Risk: High

Constraints:

- *Usability:*
  - The Use Case is self explanatory and does not require training
- *Performance:*
  - System should handle 89 requests in 1 minute, and execute them in 2 ms.

Modification History:

Owner: David Martinez

Initiation date: 09/15/2008

Date last modified: 10/06/2008

10.2.5 CVMGUI_ COM_005 - Save Communication

Details:

Actor: User

Pre-conditions:

1. There is an active communication.

Description:

1. Use case begins when the user selects Save or Save As (See Appendix C Screenshot 18).

*2.*The system prompts the user for the *Communication Name.*

3.The user specifies a name for the communication to be saved.

4.The system stores the X-CML associated with the communication in the file given by the user.

5.The system updates the main window to show the saved communication.

Post-conditions:

4.The saved communication is available on the main application window.

Alternative Courses of Action:

1.On step 1, if the current communication was previously saved and user selected "Save" steps 2 and 3 are skipped.

Related Uses Case:

1.Leave Communication

Decision Support:

Frequency: the user is expected to save communications 3 times per session.

Criticality: High.

Risk: High.

Constraints:

- *Usability:*
  - The Use Case is self explanatory, but does require 1 help document of instruction.
- *Performance:*
  - System should handle 50 requests in 1 minute, and executed them in 5 ms.

Modification History:

Owner: David Martinez

Initiation date: 09/15/2008

Date last modified: 10/06/2008

10.2.6 CVMGUI_ COM_006 –ADD PARTICIPANT

Details:

Actor: User

Pre-conditions:

2.The user has an active communication with at least one connection.

Description:

3.Use case begins when the user selects a contact from the contact list in the main application window and adds it to the connection (See Appendix C Screenshot 17).

4.The system adds the new participant to the connection.

5.The system submits the updated communication schema to the Synthesis Engine.

6.The system updates the list of participants on the connection by adding the new participant with status pending.

Post-conditions:

1.The new participant has been added to the communication.

2.The synthesis engine has received the updated communication schema.

3.The new participant appears on the participant list with status pending.

Related Use Cases:

1.Create Communication

2.Start Communication

3.Join Communication

Decision Support:

Frequency: a user is expected to add at most 6 contacts per communication, and 3 to 4 communications per day.

Criticality: high

Risk: medium

Constraints:

Usability:

- A user without experience shall be able to add a participant within 30 seconds.

Performance:

- The system shall finish adding the participant in less than 3 seconds.

Modification History:

Owner: Jorge Guerra

Initiation Date: 09/18/2008

Date Last Modified: 10/06/2008

10.2.7 CVMGUI_COM_007 - START COMMUNICATION

Details:

Actors: User, Synthesis Engine.

Pre-conditions:

4.The user has logged into the system.

5.The user has created a communication but has not started it yet.

Description:

2.Use case begins when the user chooses to start a communication.

3.The system checks if it is a valid communication, that is, it contains at least 2 participants and one connection.

4.The system submits the new communication schema to the Synthesis Engine.

5.The system indicates the user that the communication has started.

Post-conditions:

1.The Synthesis Engine has received the X-CML schema of the communication.

2.The user has been informed that the communication has started.

Alternative Courses of Action:

4.If on step 2 the communication is not valid, the system displays a message indicating it.

Related Use Cases:

2.Create Communication.

Decision Support:

Frequency: a user is expected to start a communication 5 times per application execution.

Criticality: high.

Risk: high.

Constraints:

Usability:

- A user without experience shall be able to start a new communication in less than 2 minutes.

Performance:

- The system shall finish starting a new communication schema in less than 100 milliseconds.

Modification History:

Owner: Barbara Espinoza

Initiation Date: 09/14/2008

Date Last Modified: 09/28/2008

10.2.8 CVMGUI_COM_009 – Update Communication

Details:

Actor: Synthesis Engine

Pre-conditions:

1.The user has started a communication and there is a connection already established between two or more participants.

Description:

1.Use case begins when the synthesis engine sends a negotiated schema.

2.The system finds the differences of the negotiated schema with respect to the current one.

3.The system updates the current schema with the changes from the negotiated schema.

4.The system updates the user interface to reflect the updated schema.

Post-conditions:

1.The current schema has been updated with the changes from the negotiated schema.

2.The user interface reflects the updated schema.

Related Use Cases:

1.Start Communication.

2.Add participant.

Decision Support:

Frequency: The synthesis engine is expected to update the schema 1 time per second.

Criticality: High

Risk: Medium

Constraints:

Reliability: The CVM GUI will correctly display the modifications of the schema 99% of the times

Performance: it should take less than 1 second to update the user interface after a schema is received from the Synthesis Engine.

Modification History:

Owner: Nathanael Van Vorst

Last Modified By: Barbara Espinoza

Initiation Date: 9/16/2008

Date Last Modified: 10/04/2008

10.2.9 CVMGUI_MED_001 – SHARE MEDIA

Details:

Actor: User, Synthesis Engine

Pre-conditions:

1.The user has started a communication and there is a connection already established between two or more participants.

Description:

1.Use case begins when the use case begins when the user chooses to share a particular media in a connection.

2.The system updates the communication so the media is now shared in the specified connection.

3.The system sends the updated X-CML schema to the Synthesis Engine.

4.The Synthesis Engine performs all required negotiations for sharing the selected media and sends back the negotiated X-CML schema to the system.

5.The system updates the communication schema with the changes received from the Synthesis Engine.

6.The system updates the user interface to reflect the negotiated schema.

Post-conditions:

2. The communication schema has been updated so the selected media is now being shared.

3. The Synthesis Engine has received and negotiated the updated schema.

4. The system has updated the communication schema with the schema received from the Synthesis Engine.
5. The user interface reflects the negotiated schema.
6. The selected media is now being shared by all the participants in the connection.

Exceptions:

5.On step 2 if the corresponding medium type of the selected media had not yet been added to the communication, the system adds it to the control schema.

6.On step 4 if the negotiation fails the user gets notified of the issues.

Related Use Cases:

2.Share File

3.Send Chat Message

4.Start Live Audio/Video

5.Share Generic Form

6.Share Specific Form

Decision Support:

Frequency: The user is expected to share media 20 times per minute.

Criticality: High

Risk: High

Constraints:

Reliability: Correct and accurate XCML is generated for the synthesis engine 99% of the times.

Performance: the system should finish updating the user interface in less than 2 seconds after receiving the negotiated schema from the Synthesis Engine.

Modification History:

Owner: Nathanael Van Vorst

Last Modified By: Barbara Espinoza

Initiation Date: 9/16/2008

Date Last Modified: 10/04/2008

10.2.10 CVMGUI_MED_002– Enable Live Audio/Video Medium

Details:

Pre-conditions:

2.The user has started a communication and there is at least one connection with a minimum of two active participants.

Description:

1.Use case begins when the user selects a connection.

2.The user enables the live audio/video medium on the connection (See Appendix C Screenshot 16).

3.The system updates the communication schema to reflect that change.

4.The user sends the updated communication schema to the Synthesis Engine.

5.The system updates live audio/video panel indicating that live audio/video is now enabled in the connection.

Post-conditions:

3.The live audio/video medium has been added to the connection.

4.The updated schema has been sent to the synthesis engine for negotiation.

5.The live audio/video panel indicates that live audio/video is enabled on the selected connection.

Modification History:

Owner: Nathanael Van Vorst

Initiation Date: 9/16/2008

Date Last Modified: 9/26/2008

10.2.11 CVMGUI_MED_003 – Disable Live Audio/Video Medium

Details:

Actor: User, Synthesis Engine

Pre-conditions:

1.The user has started a communication and there is at least one connection with a minimum of two active participants.

2.The live audio/video medium is enabled on the connection selected for disabling.

Description:

4.This use case begins when the user chooses to disabled the live audio/video of a connection (See Appendix C Screenshot 16).

5.The schema is updated and sent to the synthesis engine for negotiation.

6.The system updates live audio/video panel indicating that live audio/video is now disabled in the connection.

Post-conditions:

2.The live audio/video medium has been removed from the connection.

3.The updated schema has been sent to the synthesis engine for negotiation.

4.The live audio/video panel indicates that live audio/video is disabled on the selected connection.

Related Use Cases:

2.Enable Live Audio/Video Medium.

Decision Support:

Frequency: The user is expected to disable the live audio/video medium 1 time per connection and expected to have 2 connections per day.

Criticality: Low

Risk: Low

Modification History:

Owner: Nathanael Van Vorst

Initiation Date: 9/16/2008

Date Last Modified: 9/26/2008

10.2.12 CVMGUI_MED_004 – Share File

Details:

Actor: User, Synthesis Engine

Pre-conditions:

1.The user has started a communication and there is at least one connection with a minimum of two active participants.

Description:

1.Use case begins when the user selects a file and chooses to share it into a connection (See Appendix C Screenshot 15).

2.The system updates the communication so the selected file is now shared in the specified connection.

3.The system sends the updated X-CML schema to the Synthesis Engine.

4.The system updates the file panel of the selected connection by adding the selected file.

5.The Synthesis Engine performs all required negotiations for sharing the selected media and sends back the negotiated X-CML schema to the system.

6.The system updates the communication schema with the changes received from the Synthesis Engine.

7.The system updates the user interface to reflect the negotiated schema.

Post-conditions:

1.The communication schema has been updated so the selected file is now being shared.

2.The Synthesis Engine has received and negotiated the updated schema.

3.The system has updated the communication schema with the schema received from the Synthesis Engine.

4.The user interface reflects the negotiated schema.

5.The selected file is now being shared by all the participants in the connection.

Related Use Cases:

3.Share Media

Decision Support:

Frequency: The user is expected to share files 1 time every 10 minutes.

Criticality: Low

Risk: Low

Modification History:

Owner: Nathanael Van Vorst

Initiation Date: 9/16/2008

Date Last Modified: 9/26/2008

10.2.13 CVMGUI_MED_005 – Start Live Audio/Video

Details:

Actor: User, Synthesis Engine

Pre-conditions:

1.The user has started a communication and there is at least one connection with a minimum of two active participants.

2.The Live Audio/Video medium has been enabled on the connection.

Description:

4.Use case begins when the user chooses to start live audio/video on a connection (See Appendix C Screenshot 14).

5.The system updates the communication so the selected file is now shared in the specified connection.

6.The system sends the updated X-CML schema to the Synthesis Engine.

7.The system updates the file panel of the selected connection by adding the live audio/video media.

8.The Synthesis Engine performs all required negotiations for sharing starting live audio/video and sends back the negotiated X-CML schema to the system.

9.The system updates the communication schema with the changes received from the Synthesis Engine.

10.The system updates the live audio/video panel to reflect the negotiated schema.

11.The user is now receiving live audio/video from all other participants in the connection.

12.The user is now streaming live audio/video to all other participants in the connection.

Post-conditions:

1.The communication schema has been updated so live audio/video starts in the selected connection.

2.The Synthesis Engine has received and negotiated the updated schema.

3.The system has updated the communication schema with the schema received from the Synthesis Engine.

4.The user interface reflects the negotiated schema.

5.The user is now receiving live audio/video from all other participants in the connection.

6.The user is now streaming live audio/video to all other participants in the connection.

Related Use Cases:

1.Share Media

Decision Support:

Frequency: The user is expected to activate his or her live audio/video stream 1 time per connection and expected to have 2 connections per day.

Criticality: Low

Risk: Low

Modification History:

Owner: Nathanael Van Vorst

Initiation Date: 9/16/2008

Date Last Modified: 9/26/2008

10.2.14 CVMGUI_MED_006 – SEND CHAT MESSAGE

Details:

Actor: User, Synthesis Engine

Pre-conditions:

6.The user has started a communication and there is at least one connection with a minimum of two active participants.

Description:

1.The user sends a chat message (See Appendix C Screenshot 13).

2.The system updates the communication so the chat message is now shared in the specified connection.

3.The system sends the updated X-CML schema to the Synthesis Engine.

4.The system updates the chat panel of the selected connection by adding the chat message.

5.The Synthesis Engine performs all required negotiations for sending the chat message and sends back the negotiated X-CML schema to the system.

6.The system updates the communication schema with the changes received from the Synthesis Engine.

Post-conditions:

1.The communication schema has been updated so the chat message gets sent in the selected connection.

2.The Synthesis Engine has received and negotiated the updated schema.

3.The system has updated the communication schema with the schema received from the Synthesis Engine.

4.The chat message sent appears on the chat panel.

Related Use Cases:

1.Share Media

Decision Support:

Frequency: The user is expected to send 5 chat message per second.

Criticality: Low

Risk: Low

Modification History:

Owner: Nathanael Van Vorst

Initiation Date: 9/16/2008

Date Last Modified: 9/26/2008

10.2.15 CVMGUI_MED_007 – Create Generic Form

Details:

Actor: User

Pre-conditions:

3.The user has logged in.

Description:

1.This use case begins when the use begins when the user selects to *Create new generic form* (See Appendix C Screenshot 8).

2.The system displays a dialog asking for which files are to be included in the form.

3.The user selects the files he/she wishes the add to the generic form, and specifies a name for it.

4.The system creates a new generic form with the files indicated by the user and displays it in the appropriate panel.

Post-conditions:

1.The new form is displayed in the appropriate panel.

Alternative Courses of Action:

1.In step 3 the user may choose to cancel the creation for the form.

Exceptions:

1.In step 3 if the user does not add files to the form, the system tells him/her that it can not create empty forms.

2.In step 3 if the user does not specify a name for the form the system, will re ask for a name.

Related Use Cases:

3.Share Generic Form

Decision Support:

Frequency: the user is expected to create generic forms twice per connection.

Criticality: low

Risk: low

Constraints

Usability:

- A user without experience shall be able to create a generic form within 1 minute.

Performance:

- The system shall finish creating a new generic form in 10 milliseconds times the number of files added.

Modification History:

Owner: Jorge Guerra

Initiation Date: 10/06/2008

Date Last Modified: 10/06/2008

10.2.16 CVMGUI_MED_008 – Share Generic Form

Details:

Actor: User, Synthesis Engine

Pre-conditions:

1.The user has an active connection.

2.Some generic forms have already been created.

Description:

1.The use begins when the user selects a generic form and adds it to a connection (See Appendix C Screenshot 11).

2.The system updates the communication so that the given generic form is now shared in the specified connection.

3.The system sends the updated X-CML schema to the Synthesis Engine.

4.The Synthesis Engine performs all required negotiations for sharing the selected generic form and sends back the resulting X-CML schema to the system.

5.The system updates the user interface to reflect the negotiated schema.

Post-conditions:

1.The communication schema has been updated to indicate that the selected form is now being shared.

2.The user interface reflects the negotiated schema.

3.The selected generic form is now being shared among all participants of the connection.

Exceptions:

1.In step 3, if the user does not add files to the form, the system tells him/her that it cannot create empty forms.

2.In step 4, if the negotiation fails the user gets notified of the issue.

Related Use Cases:

1.Create Generic Form

2.Share Specific Form

3.Share Media

Decision Support:

Frequency: the user is expected to share generic forms four times per connection.

Criticality: low

Risk: low

Constraints

Usability:

- A user without experience shall be able to share a generic form within 30 seconds.

Performance:

- The system shall finish sharing a generic form in 100 milliseconds times the number of files in the form.

Modification History:

Owner: Jorge Guerra

Initiation Date: 10/06/2008

Date Last Modified: 10/06/2008

10.2.17 CVMGUI_MED_009 – Share Specific Form

Details:

Actor: User, Mediator, and Synthesis Engine

Pre-conditions:

1.The user has an active connection.

2.Some specific forms exist.

Description:

1.The use case begins when the user selects a specific form and adds it to a connection. (See Appendix C Screenshot 9 and Screenshot 11).

2.The system updates the communication so that the given specific form is now shared in the specified connection.

3.The system sends the updated X-CML schema to the Synthesis Engine.

4.The Synthesis Engine performs all required negotiations for sharing the selected specific form and sends back the resulting X-CML schema to the system.

5.The system updates the user interface to reflect the negotiated schema.

Post-conditions:

1. The communication schema has been updated to indicate that the selected form is now being shared.
2. The user interface reflects the negotiated schema.
3. The selected specific form is now being shared among all participants of the connection.

Exceptions:

1.In step 3, if the user does not add files to the form, the system tells him/her that it cannot create empty forms.

2.In step 4, if the negotiation fails the user gets notified of the issue.

Related Use Cases:

1.Share Generic Form

2.Share Media

Decision Support:

Frequency: the user is expected to share specific forms twice per connection.

Criticality: low

Risk: low

Constraints

Usability:

- A user without experience shall be able to share an specific form within 30 seconds.

Performance:

- The system shall finish sharing a specific form in 100 milliseconds times the number of files in the form.

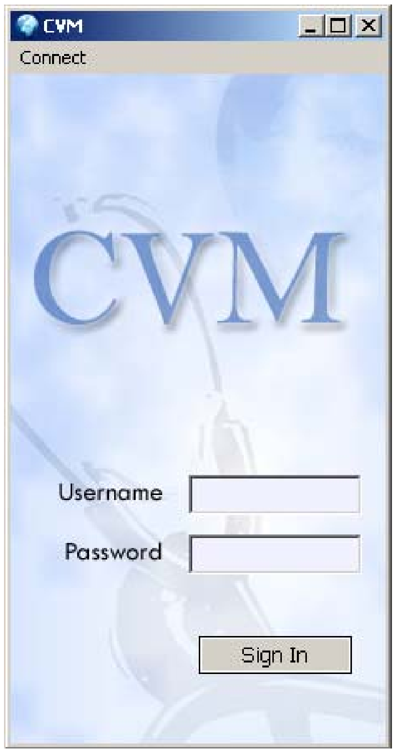
Modification History:

Owner: Jorge Guerra

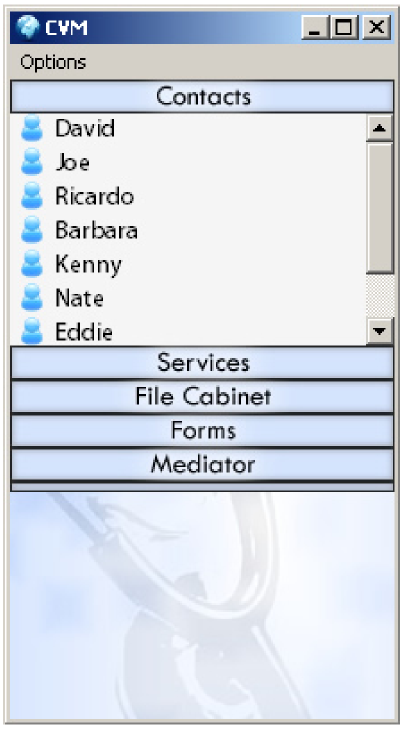
Initiation Date: 10/06/2008

Date Last Modified: 10/06/2008

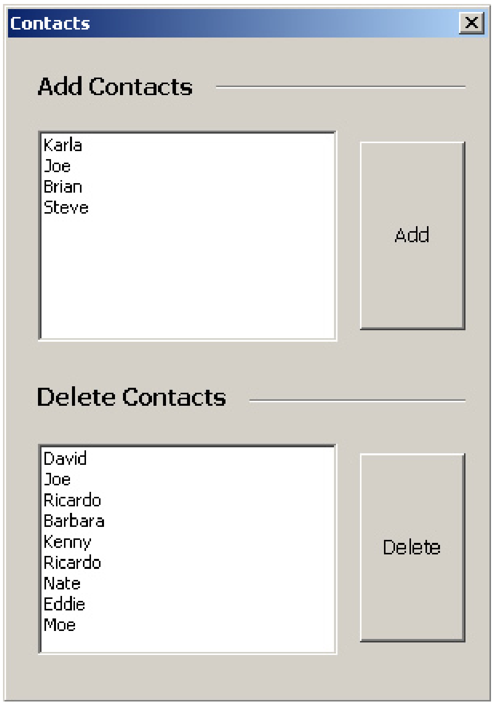
10.3 Appendix C – User Interface



Screenshot 1. *Log In*



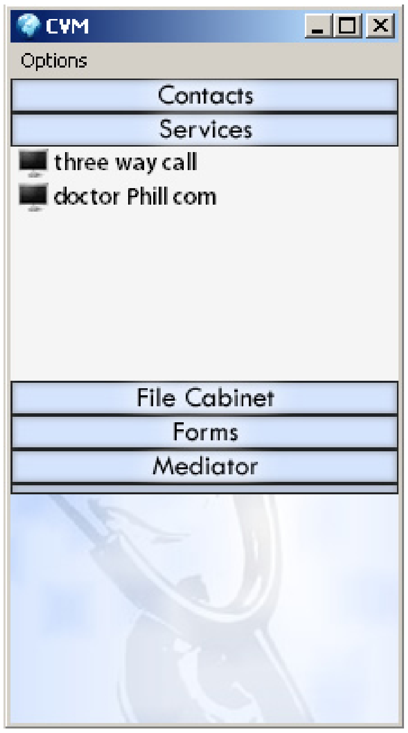
***Screenshot 2**. Contacts Tab*



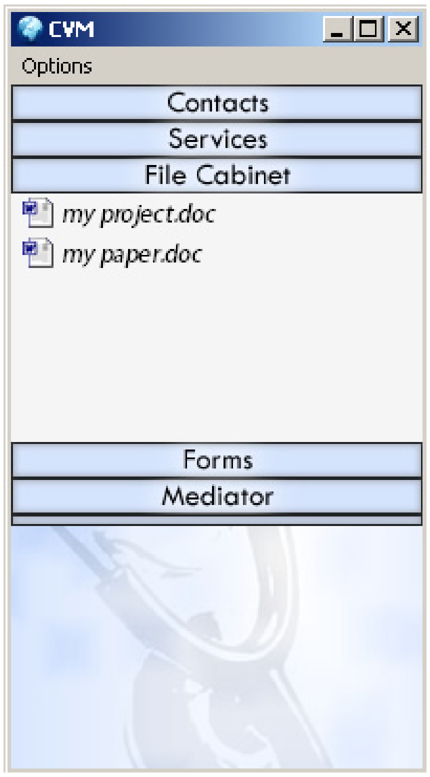
Karla
Joe
Brian
Steve

David
Joe
Ricardo
Barbara
Kenny
Ricardo
Nate
Eddie
Moe

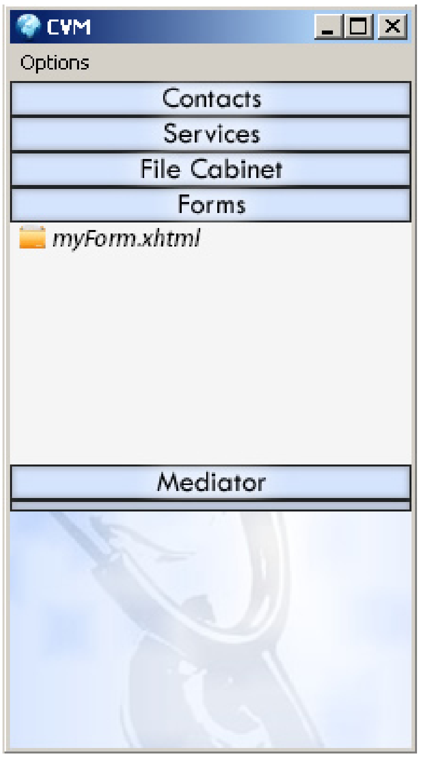
***Screenshot 3**. Add/Delete Contacts*



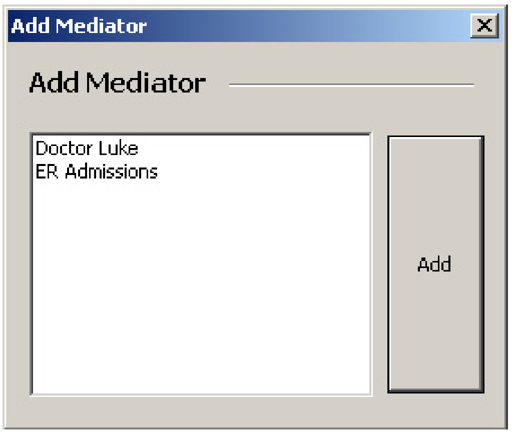
***Screenshot 4**. Communication Services Tab*



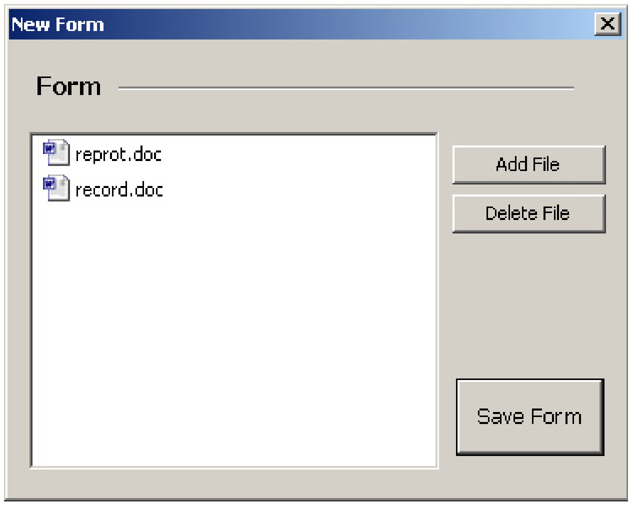
Screenshot 5*. File Cabinet Tab*



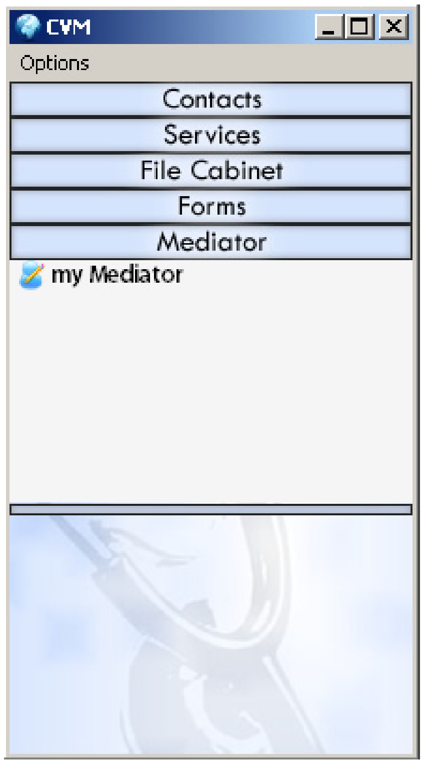
Screenshot 6. *Forms Tab*



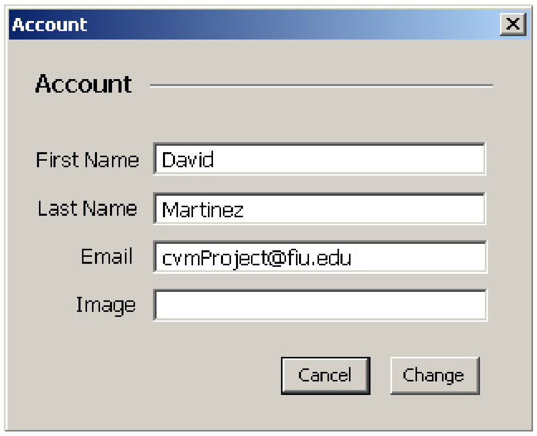
Screenshot 7. *Add Mediator*



Screenshot 8. *Create New/Edit Generic Form*



Screenshot 9. *Mediator Window*



Account

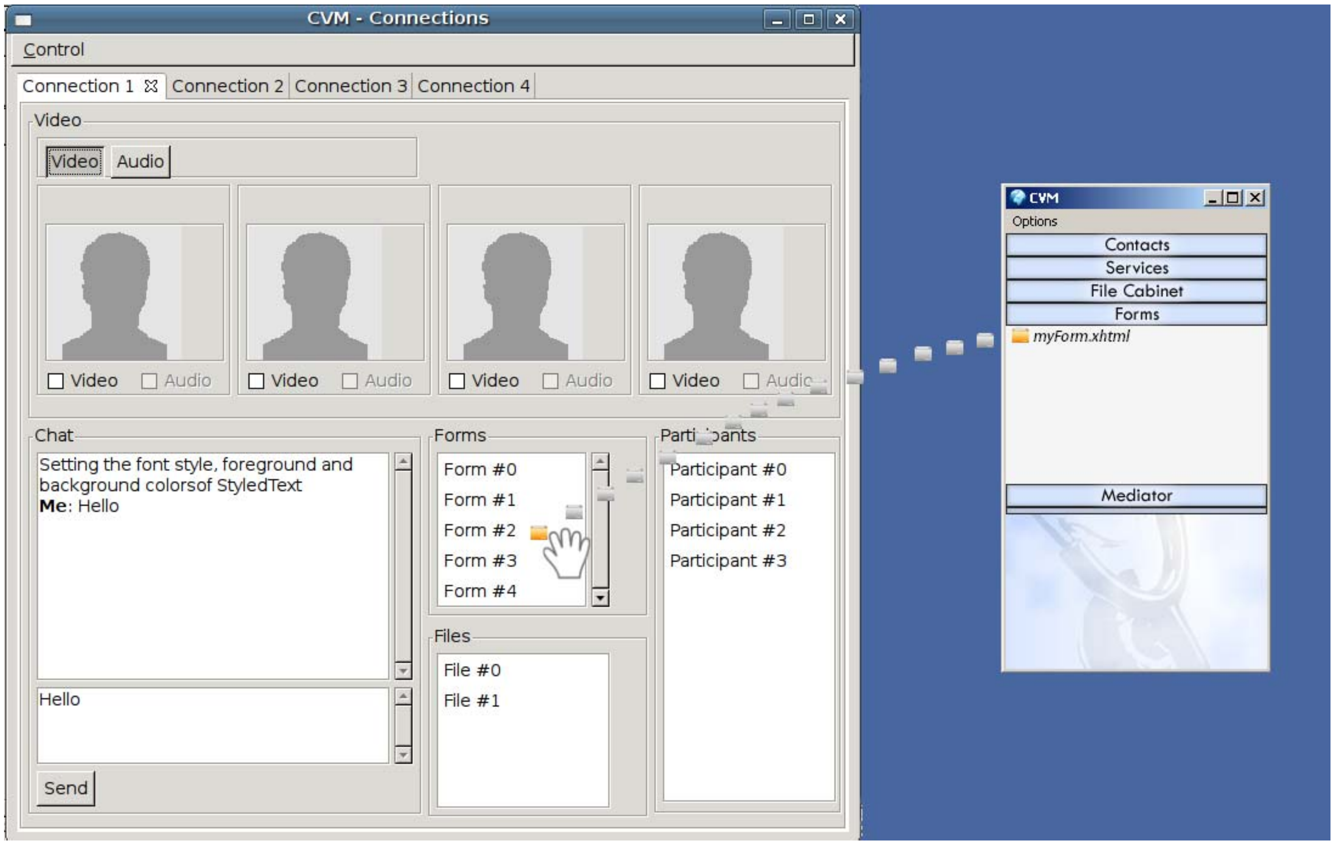
First Name David

Last Name Martinez

Email cvmProject@fiu.edu

Image

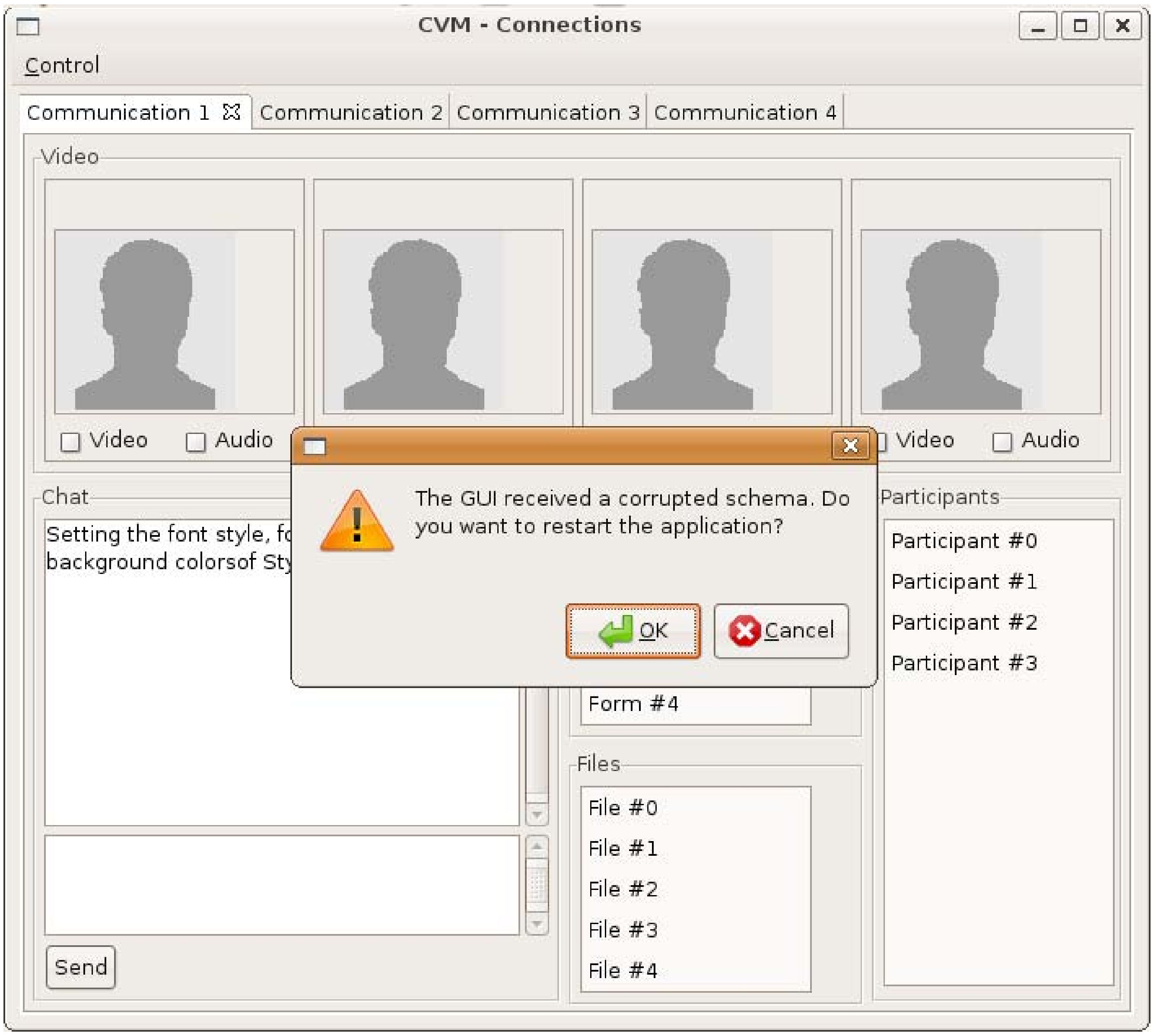
Screenshot 10. *Edit Account*



Setting the font style, foreground and background colorsof StyledText
Me: Hello

Hello

Screenshot 11. Share Specific/Generic Form



The GUI received a corrupted schema. Do you want to restart the application?

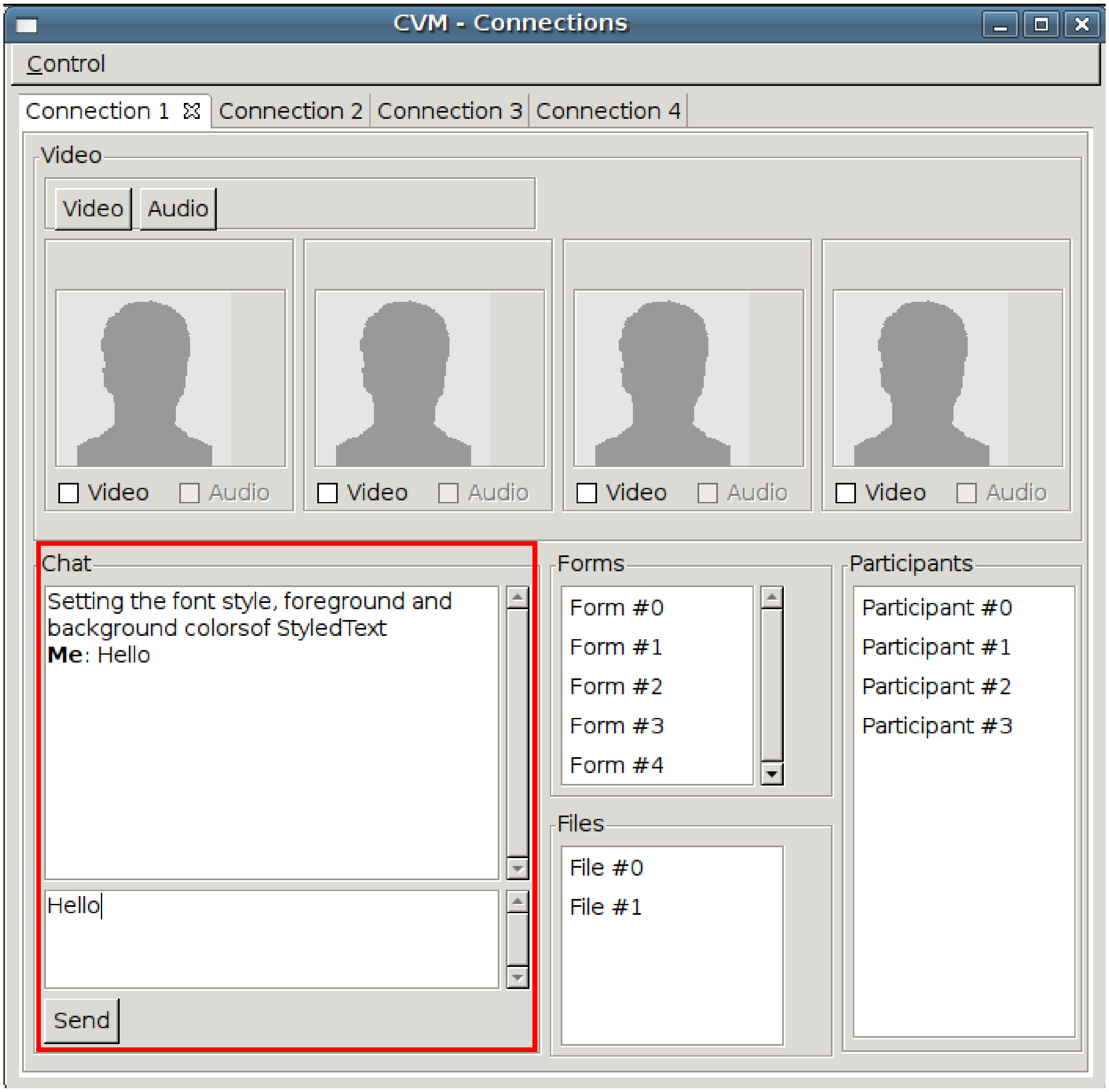
Setting the font style, fo
background colorsof Sty

Participant #0
Participant #1
Participant #2
Participant #3

Form #4

File #0
File #1
File #2
File #3
File #4

Screenshot 12. *Validate Communication*



Setting the font style, foreground and background colorsof StyledText
Me: Hello

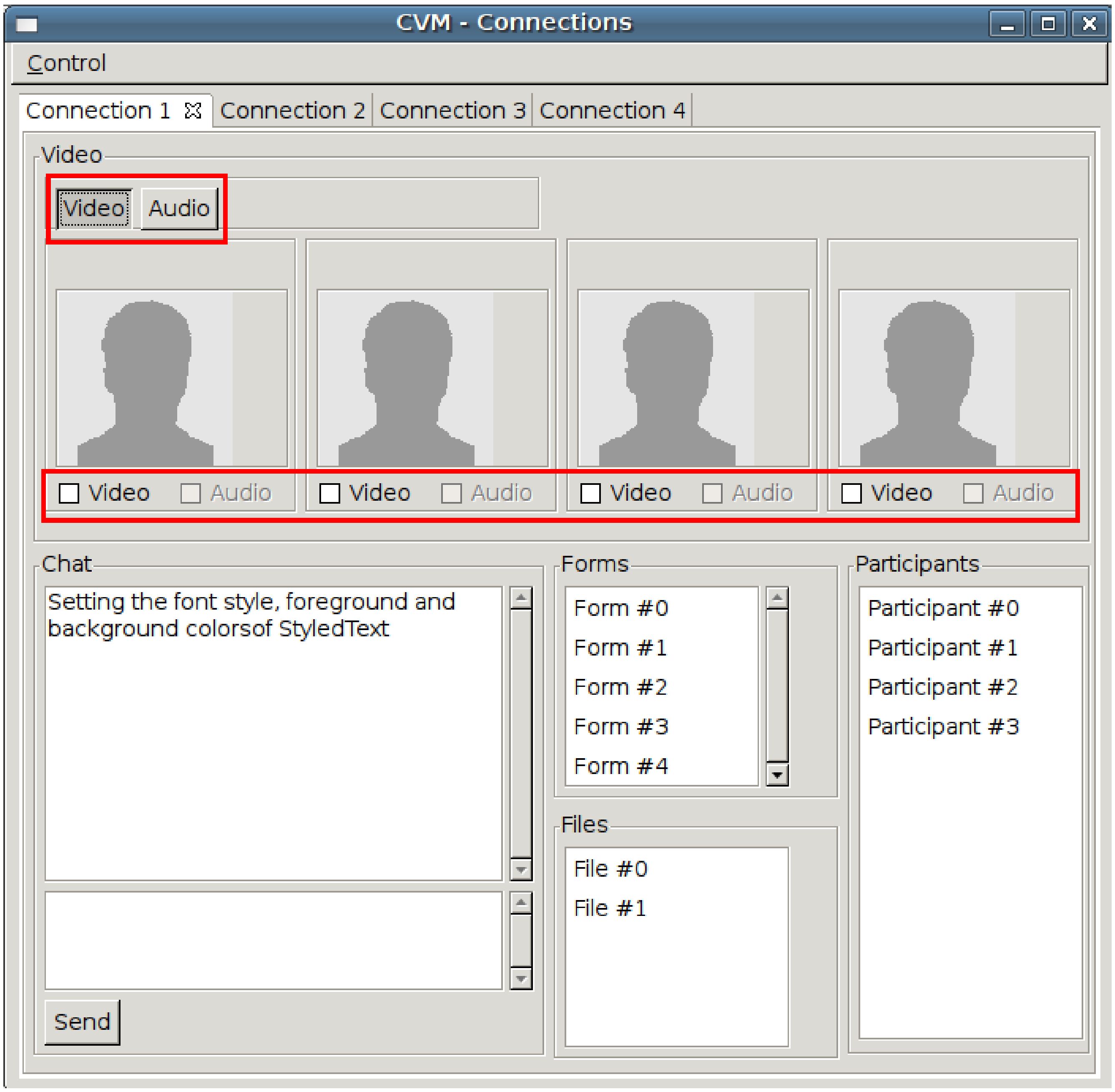
Hello

Form #0
Form #1
Form #2
Form #3
Form #4

File #0
File #1

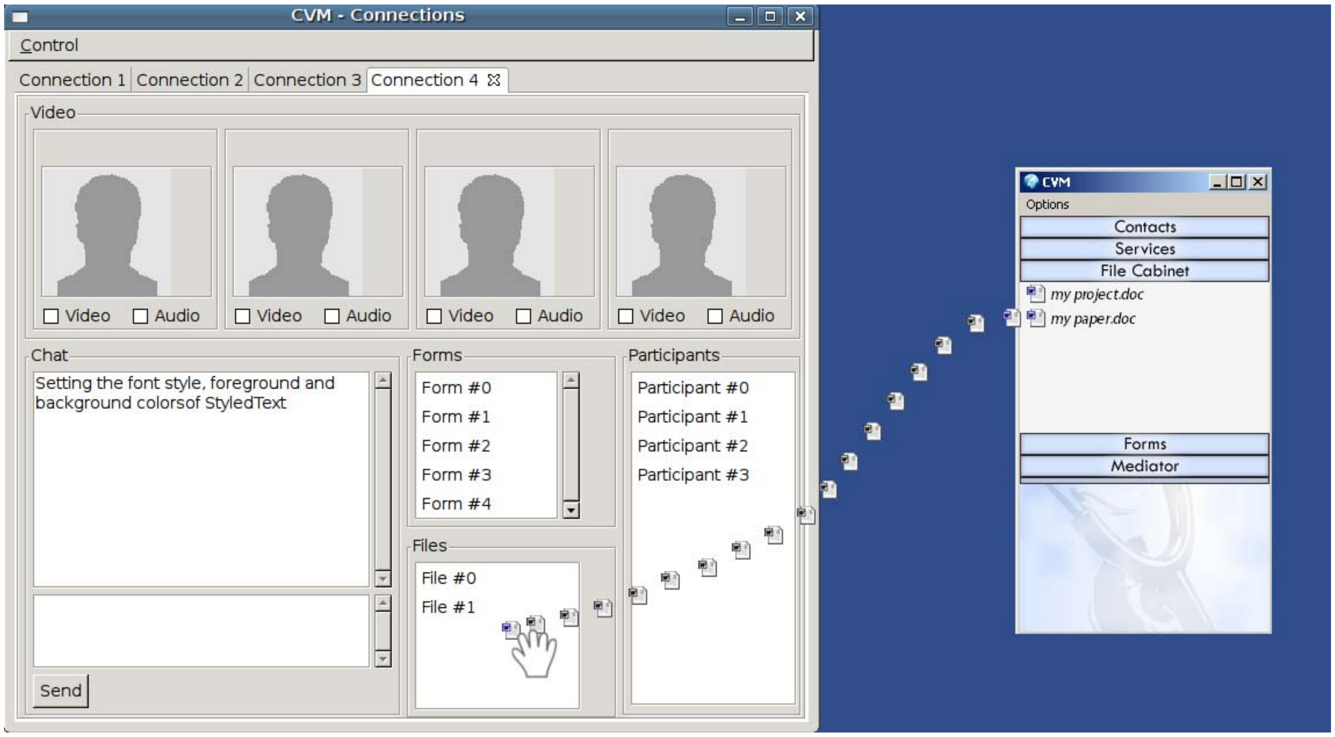
Participant #0
Participant #1
Participant #2
Participant #3

Screenshot 13. *Send Chat Message*



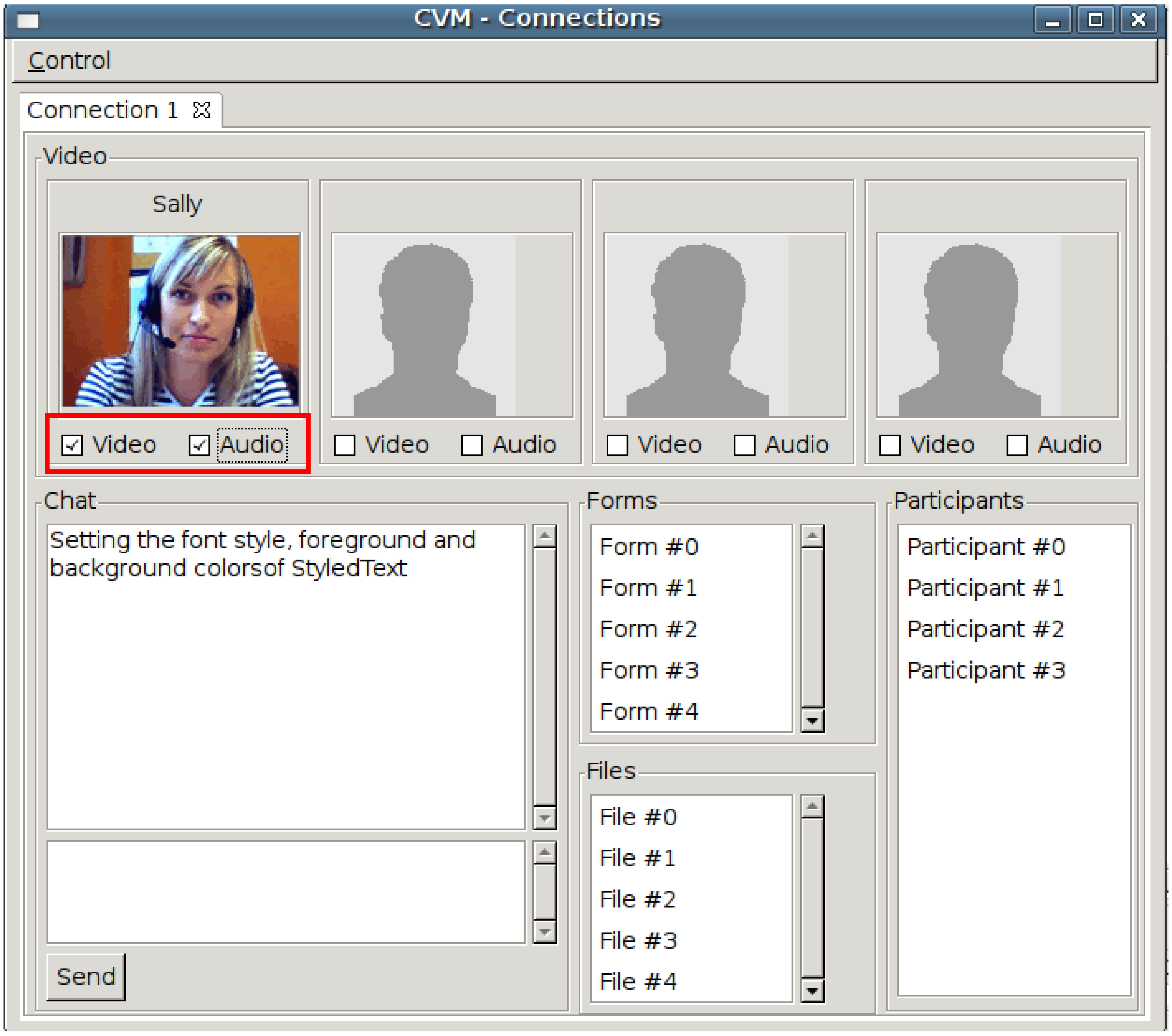
Setting the font style, foreground and background colorsof StyledText

Screenshot 14. *Start Live Audio/Video Medium*



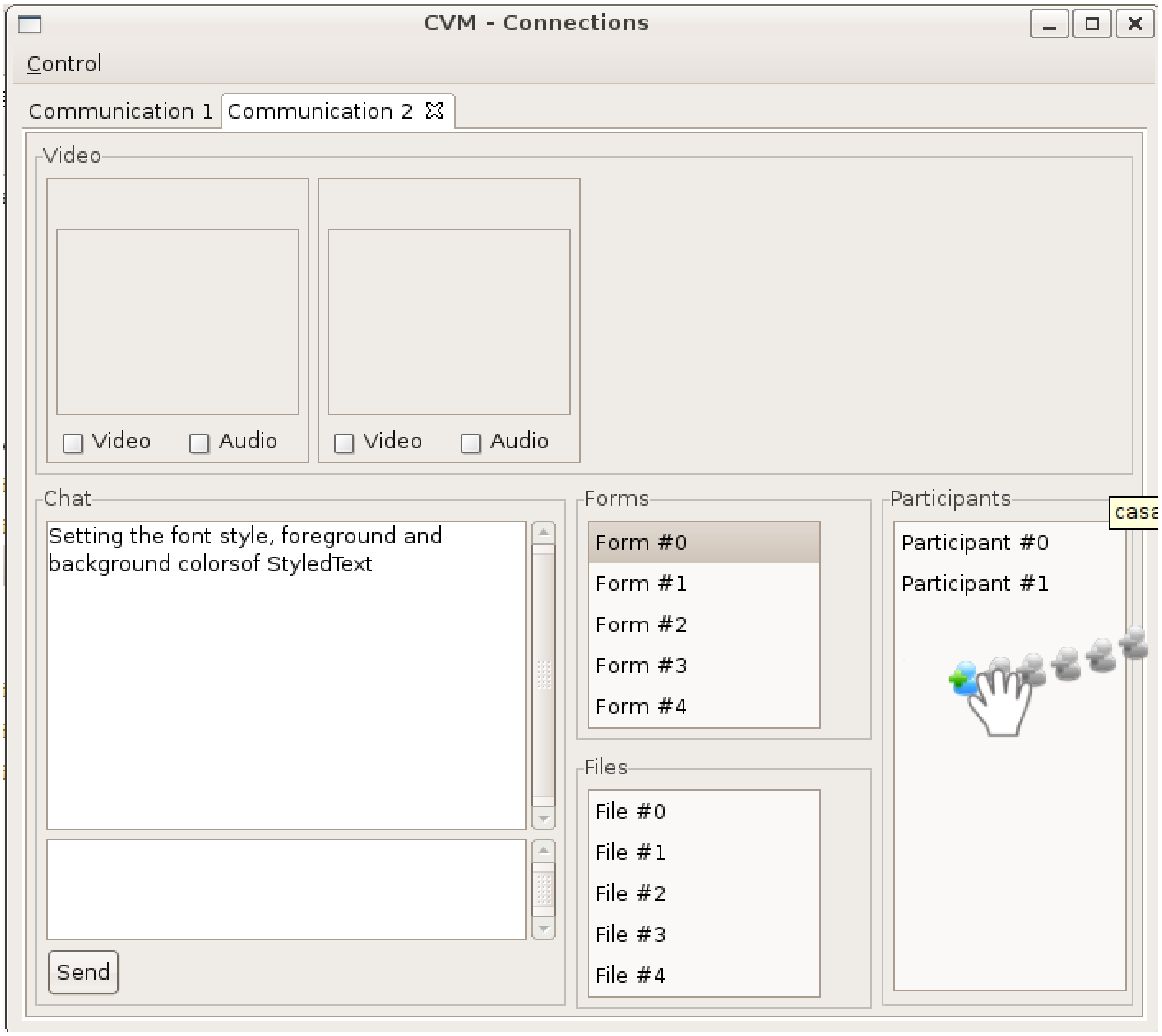
Setting the font style, foreground and background colorsof StyledText

Screenshot 15. *Share File*



Setting the font style, foreground and background colorsof StyledText

Screenshot 16. *Enable/Disable Live Audio/Video Medium*



Setting the font style, foreground and background colorsof StyledText

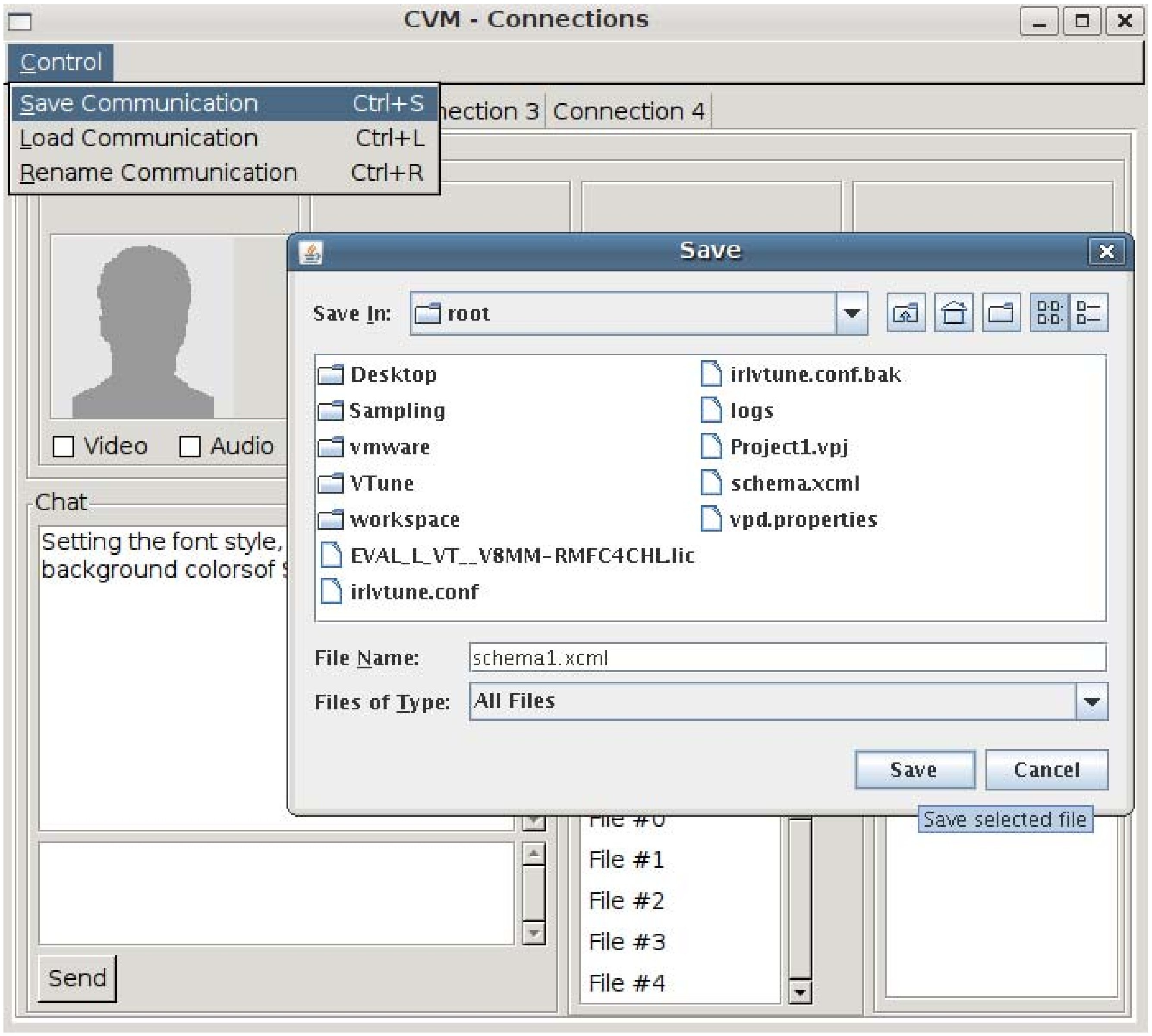
Form #0
Form #1
Form #2
Form #3
Form #4

Files

File #0
File #1
File #2
File #3
File #4

Participant #0
Participant #1

Screenshot 17. *Add Participant*

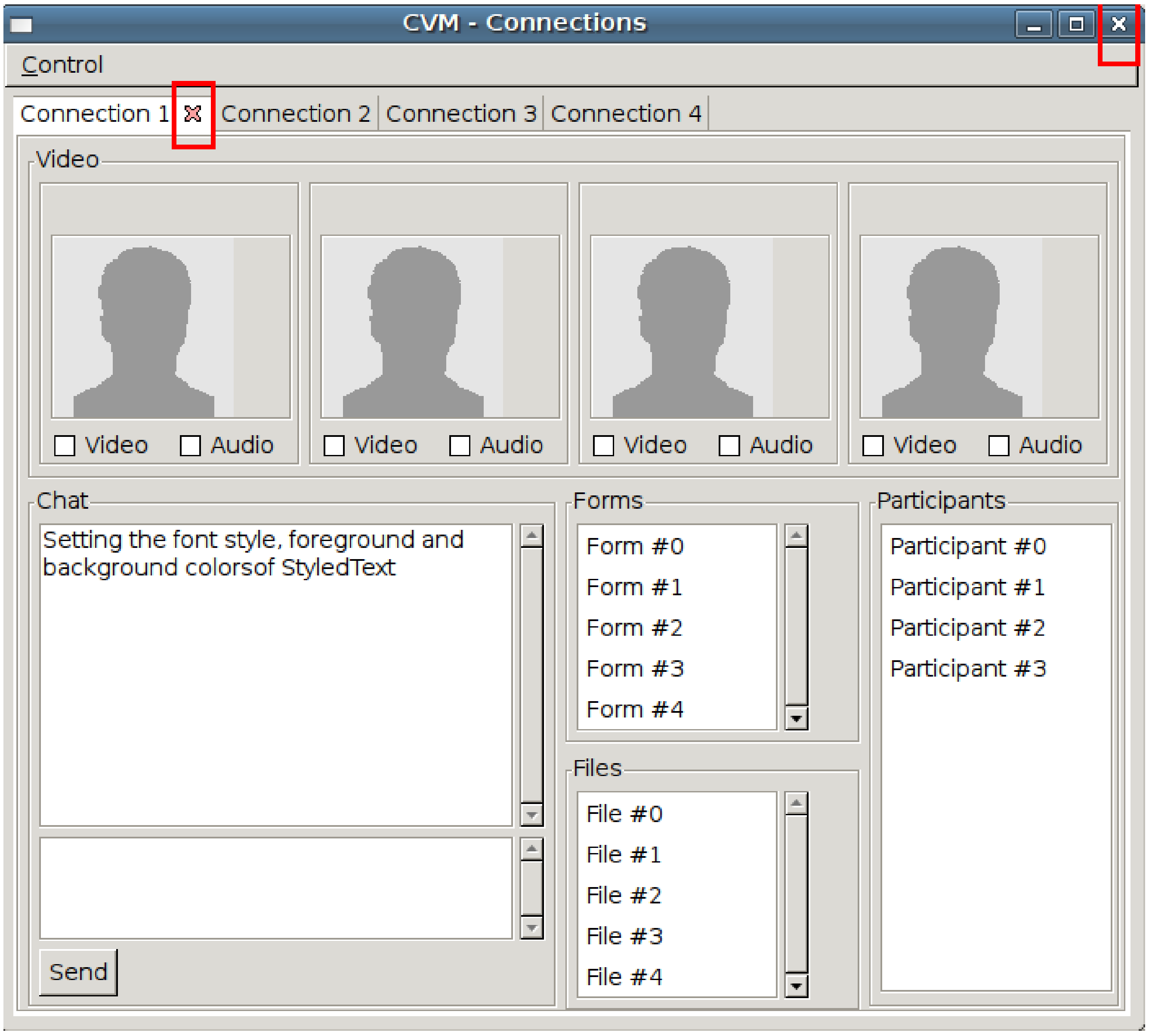


Setting the font style,
background colorsof

File Name: schema1.xcml

Files of Type: All Files

Screenshot 18. *Save Communication*



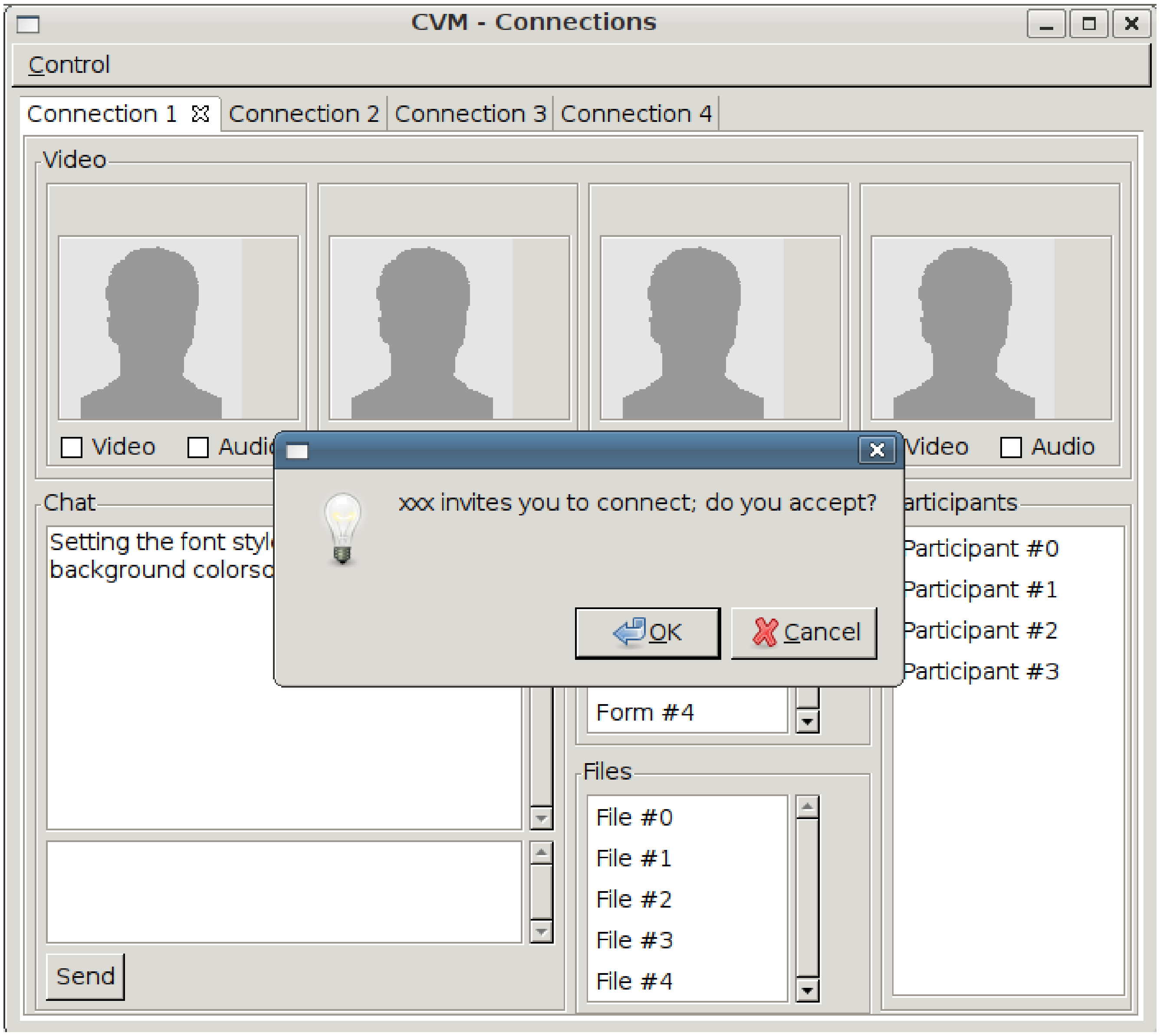
Setting the font style, foreground and background colorsof StyledText

Form #0
Form #1
Form #2
Form #3
Form #4

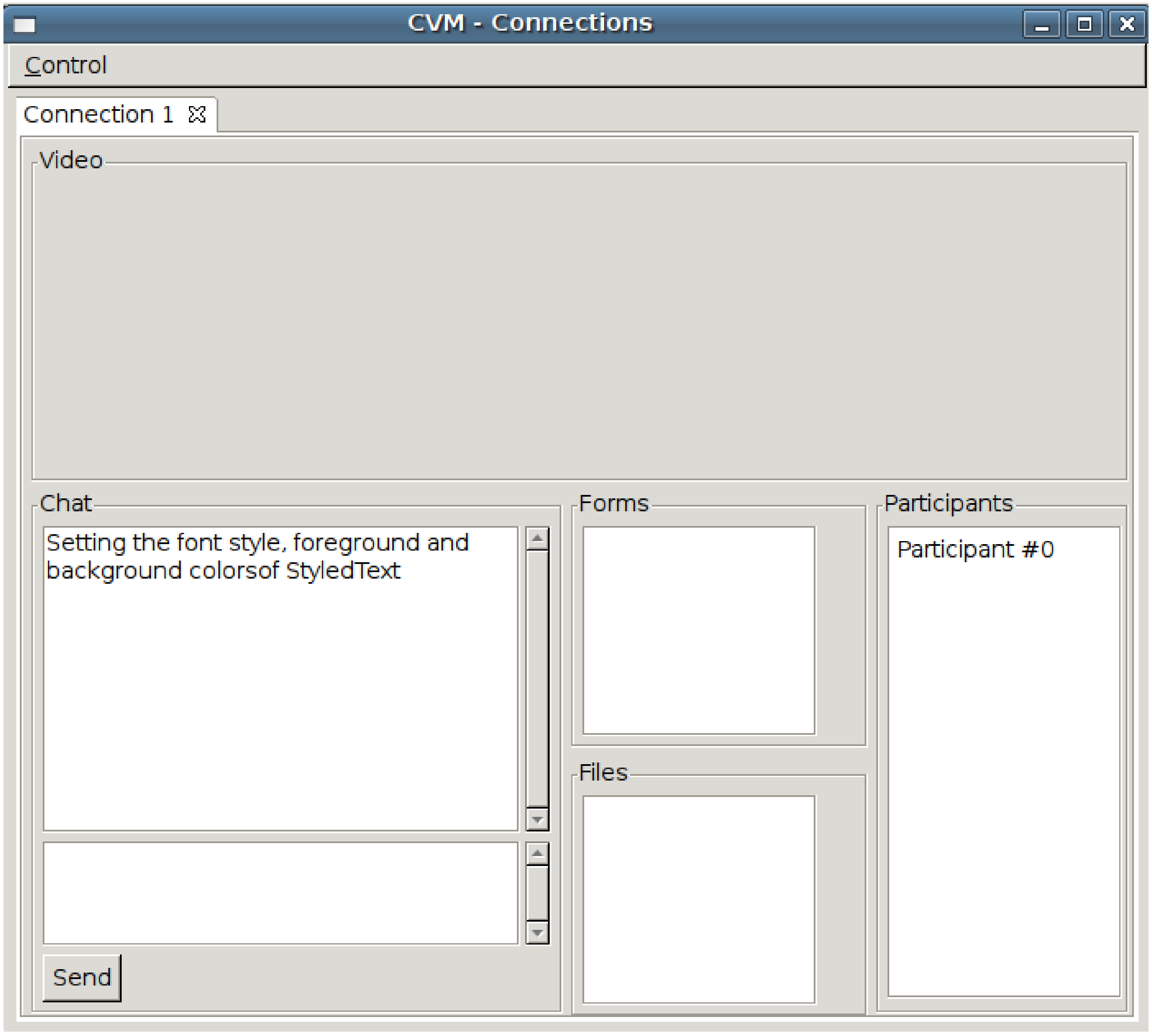
File #0
File #1
File #2
File #3
File #4

Participant #0
Participant #1
Participant #2
Participant #3

Screenshot 19. *Leave Communication*



Screenshot 20. *Join Communication*



Setting the font style, foreground and background colorsof StyledText

Participant #0

Screenshot 21. *Create Communication*

10.4 Appendix D – Detailed Class Diagrams

This section presents the detailed class diagrams for each of the packages in the CVM GUI.

10.4.1 GUI Package

The class diagram below illustrates the relationships between the classes in the GUI Package.

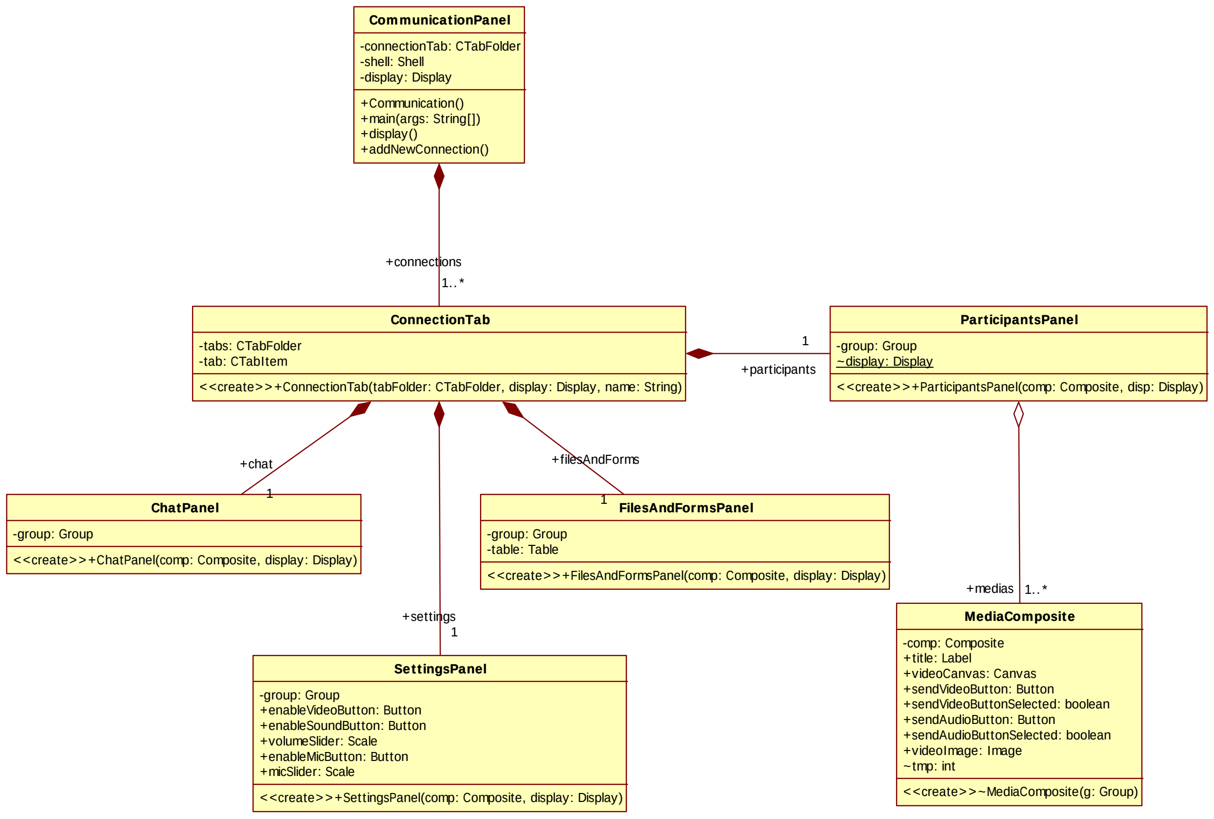


Figure 44 GUI Package Class Diagram

10.4.2 GUI.TAB_PANEL Package

The class diagram below illustrates the relationships between the classes in the GUI.TAB_PANEL Package.

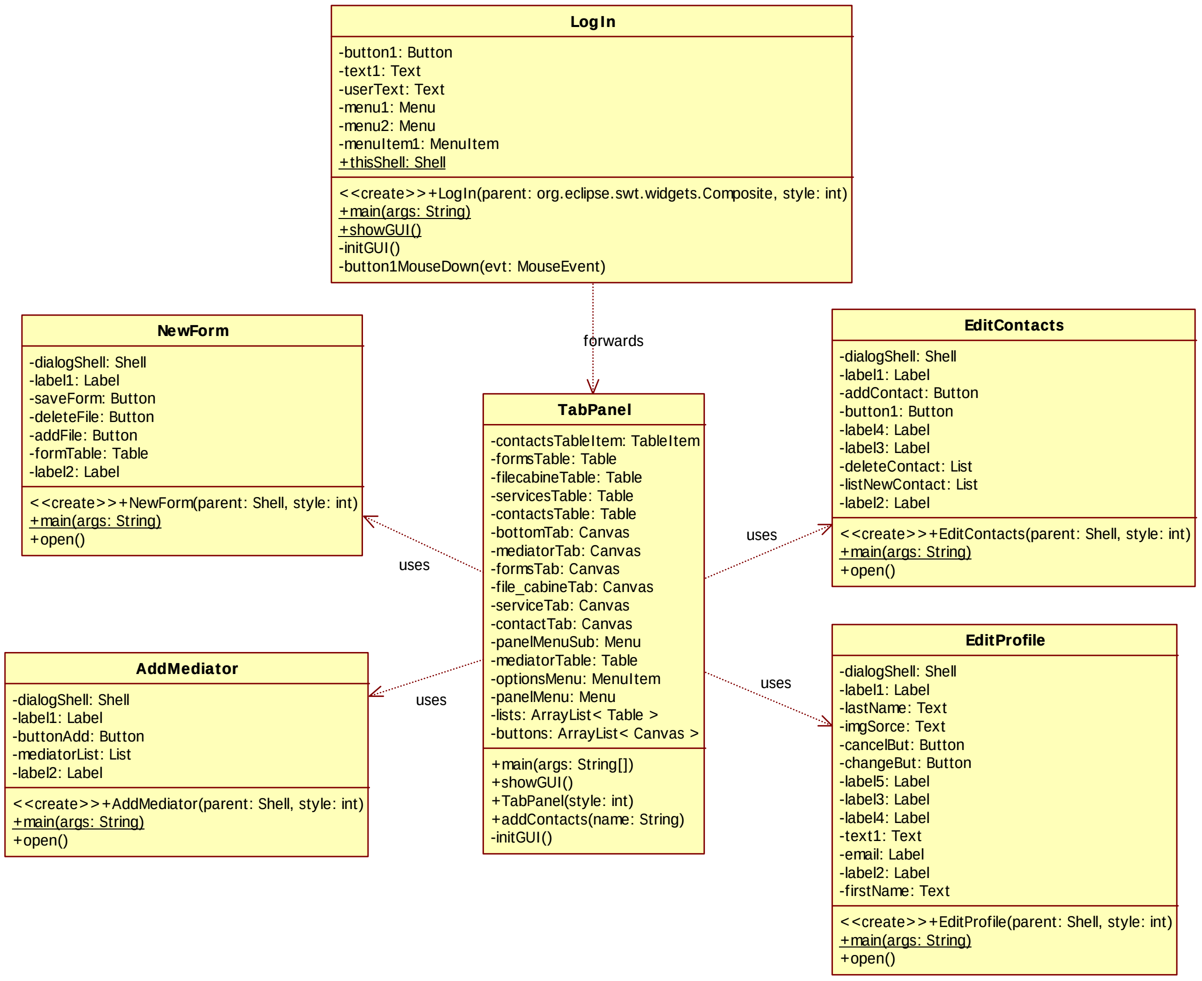


Figure 45 GUI.TAB_PANEL Package Class Diagram

10.4.3 GUI.CONTROL PACKAGE

The class diagram below illustrates the relationships between the classes in the GUI.CONTROL Package.

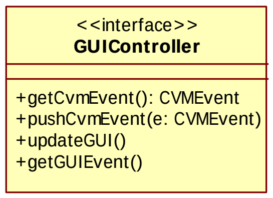


Figure 46 GUI.CONTROL Package Class Diagram

10.4.4 XCML Package

The class diagrams below illustrates the relationships between the classes in the XCML Package.

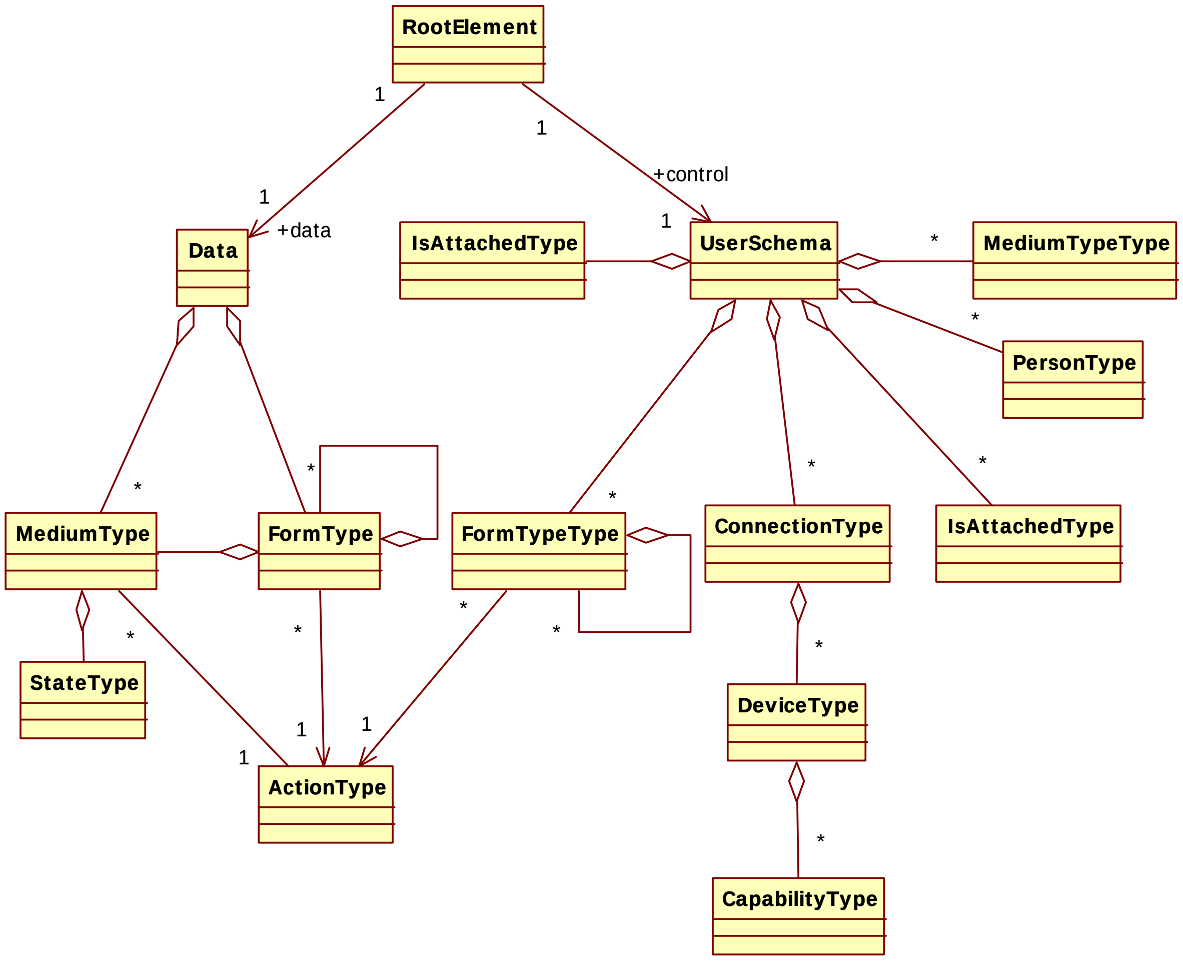


Figure 47 XCML Package Class Diagram

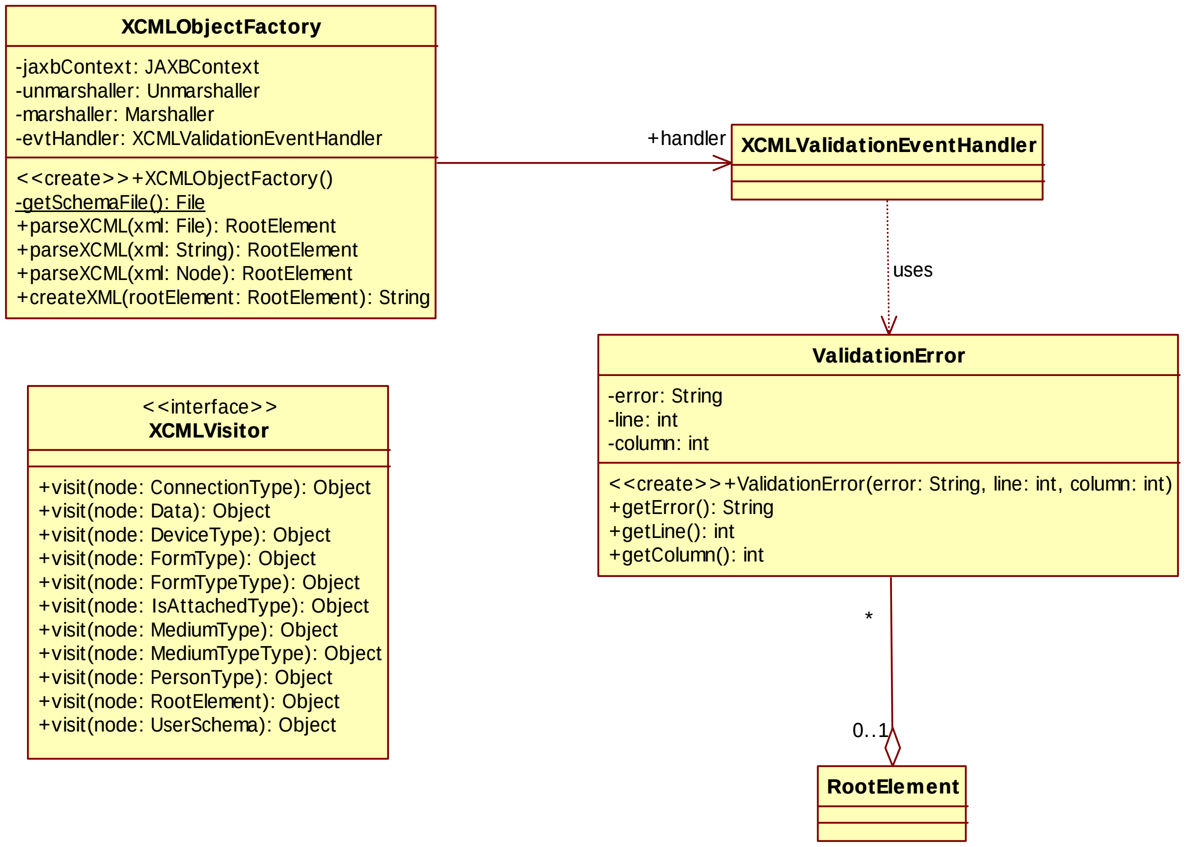

XCMLObjectFactory
-jaxbContext: JAXBContext
-unmarshaller: Unmarshaller
-marshaller: Marshaller
-evtHandler: XCMLValidationEventHandler
<<create>>+XCMLObjectFactory()
-getSchemaFile(): File
+parseXCML(xml: File): RootElement
+parseXCML(xml: String): RootElement
+parseXCML(xml: Node): RootElement
+createXML(rootElement: RootElement): String
+handler
XCMLValidationEventHandler
uses
ValidationError
-error: String
-line: int
-column: int
<<create>>+ValidationError(error: String, line: int, column: int)
+getError(): String
+getLine(): int
+getColumn(): int
*
0..1
RootElement
<<interface>>
XCMLVisitor
+visit(node: ConnectionType): Object
+visit(node: Data): Object
+visit(node: DeviceType): Object
+visit(node: FormType): Object
+visit(node: FormTypeType): Object
+visit(node: IsAttachedType): Object
+visit(node: MediumType): Object
+visit(node: MediumTypeType): Object
+visit(node: PersonType): Object
+visit(node: RootElement): Object
+visit(node: UserSchema): Object

Figure 48 XCML Package Class Diagram II

10.4.5 UCI and UCI.IMPL Packages

The class diagram below illustrates the relationships between the classes in the UCI.IMPL Package.

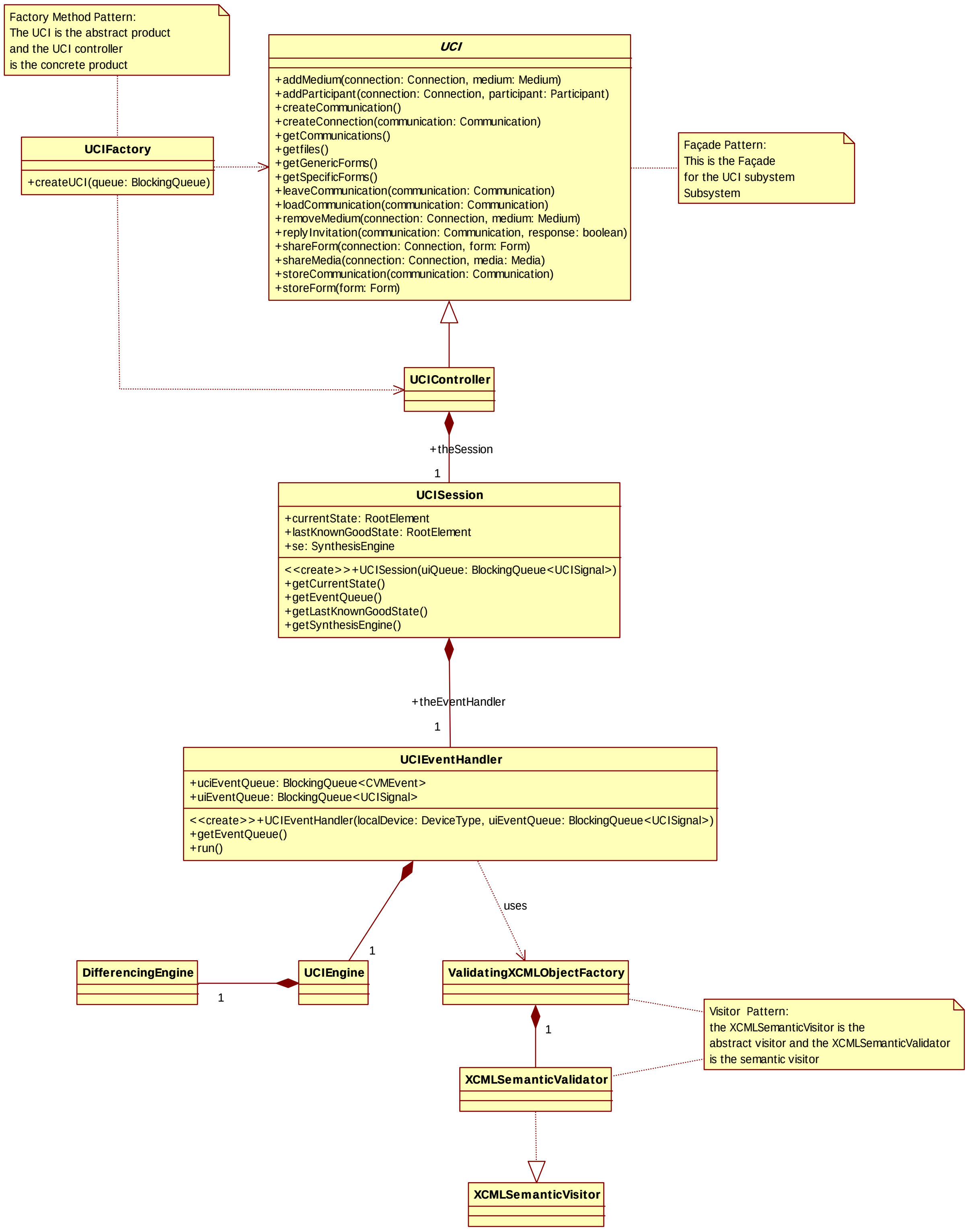

Factory Method Pattern:
The UCI is the abstract product
and the UCI controller
is the concrete product
UCI
+addMedium(connection: Connection, medium: Medium)
+addParticipant(connection: Connection, participant: Participant)
+createCommunication()
+createConnection(communication: Communication)
+getCommunications()
+getfiles()
+getGenericForms()
+getSpecificForms()
+leaveCommunication(communication: Communication)
+loadCommunication(communication: Communication)
+removeMedium(connection: Connection, medium: Medium)
+replyInvitation(communication: Communication, response: boolean)
+shareForm(connection: Connection, form: Form)
+shareMedia(connection: Connection, media: Media)
+storeCommunication(communication: Communication)
+storeForm(form: Form)
Façade Pattern:
This is the Façade
for the UCI subystem
Subsystem
UCIFactory
+createUCI(queue: BlockingQueue)
UCIController
+theSession
1
UCISession
+currentState: RootElement
+lastKnownGoodState: RootElement
+se: SynthesisEngine
<<create>>+UCISession(uiQueue: BlockingQueue<UCISignal>)
+getCurrentState()
+getEventQueue()
+getLastKnownGoodState()
+getSynthesisEngine()
+theEventHandler
1
UCIEventHandler
+uciEventQueue: BlockingQueue<CVMEvent>
+uiEventQueue: BlockingQueue<UCISignal>
<<create>>+UCIEventHandler(localDevice: DeviceType, uiEventQueue: BlockingQueue<UCISignal>)
+getEventQueue()
+run()
uses
1
DifferencingEngine
1
UCIEngine
ValidatingXCMLObjectFactory
1
Visitor Pattern:
the XCMLSemanticVisitor is the
abstract visitor and the XCMLSemanticValidator
is the semantic visitor
XCMLSemanticValidator
XCMLSemanticVisitor

Figure 49 UCI and UCI.IMPL Packages Class Diagram

10.4.6 UCI.SIGNAL PACKAGE

The class diagram below illustrates the relationships between the classes in the UCI.SIGNAL Package.

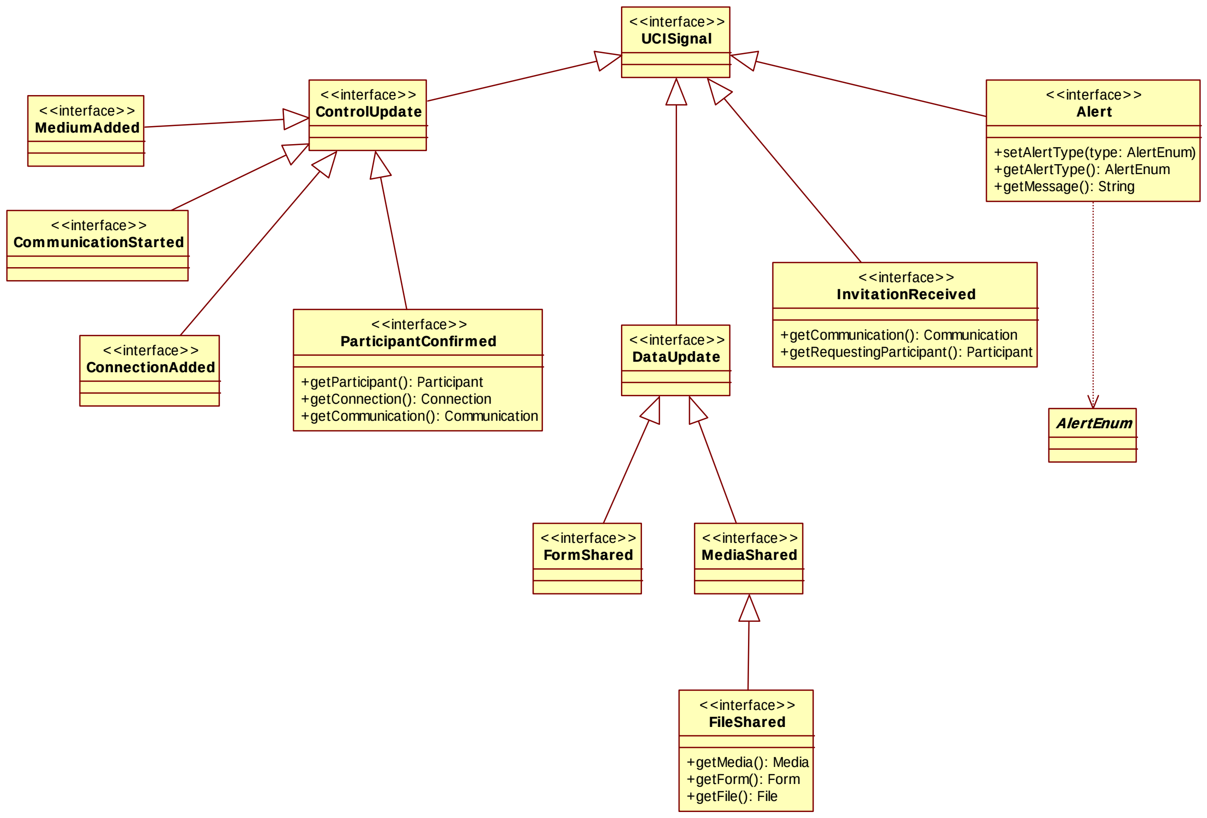

<<interface>> UCISignal
<<interface>> MediumAdded
<<interface>> ControlUpdate
<<interface>> Alert
+setAlertType(type: AlertEnum)
+getAlertType(): AlertEnum
+getMessage(): String
<<interface>> CommunicationStarted
<<interface>> InvitationReceived
+getCommunication(): Communication
+getRequestingParticipant(): Participant
<<interface>> ConnectionAdded
<<interface>> ParticipantConfirmed
+getParticipant(): Participant
+getConnection(): Connection
+getCommunication(): Communication
<<interface>> DataUpdate
AlertEnum
<<interface>> FormShared
<<interface>> MediaShared
<<interface>> FileShared
+getMedia(): Media
+getForm(): Form
+getFile(): File

Figure 50 UCI.SIGNAL Class Diagram

10.4.7 UCI.IMPL.REQUESTS PACKAGE

The class diagram below illustrates the relationships between the classes in the UCI.IMPL.REQUESTS Package.

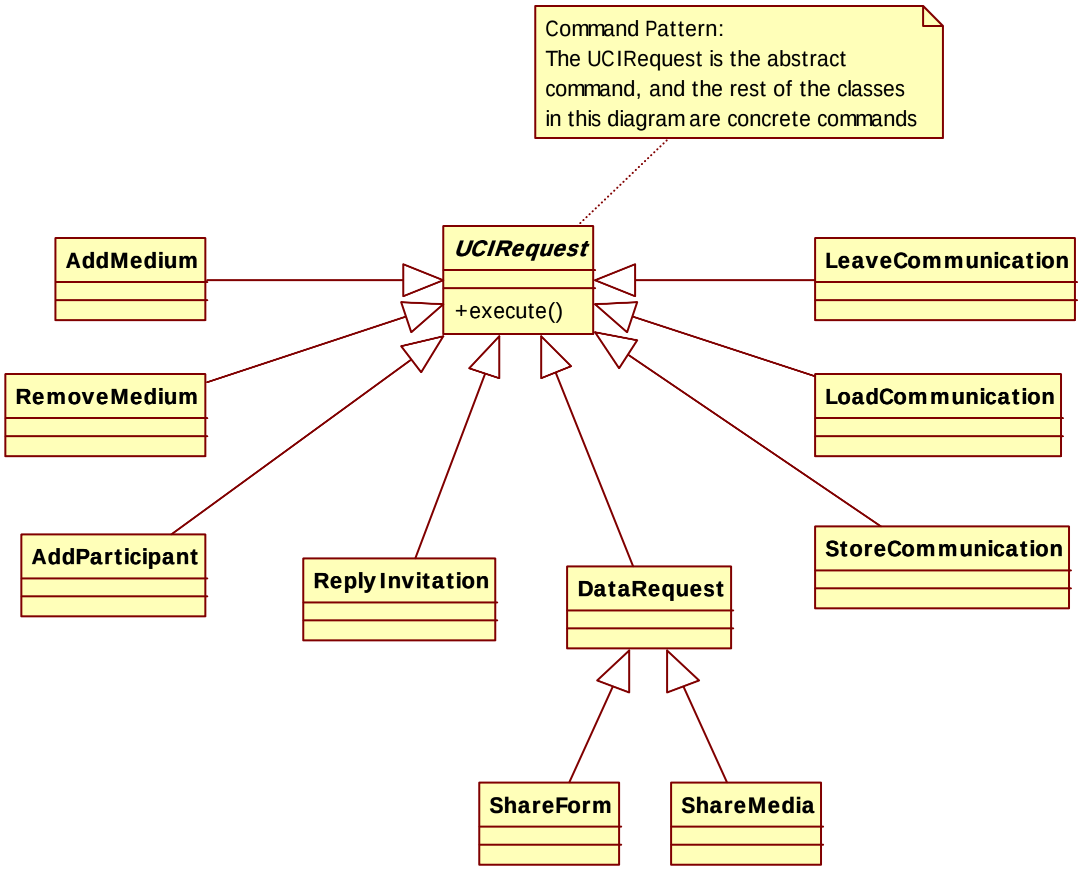


Figure 51 UCI.IMPL.REQUESTS Package Class Diagram

10.4.8 UCI.COMPONENT PACKAGE

The class diagram below illustrates the relationships between the classes in the UCI.COMPONENT Package.

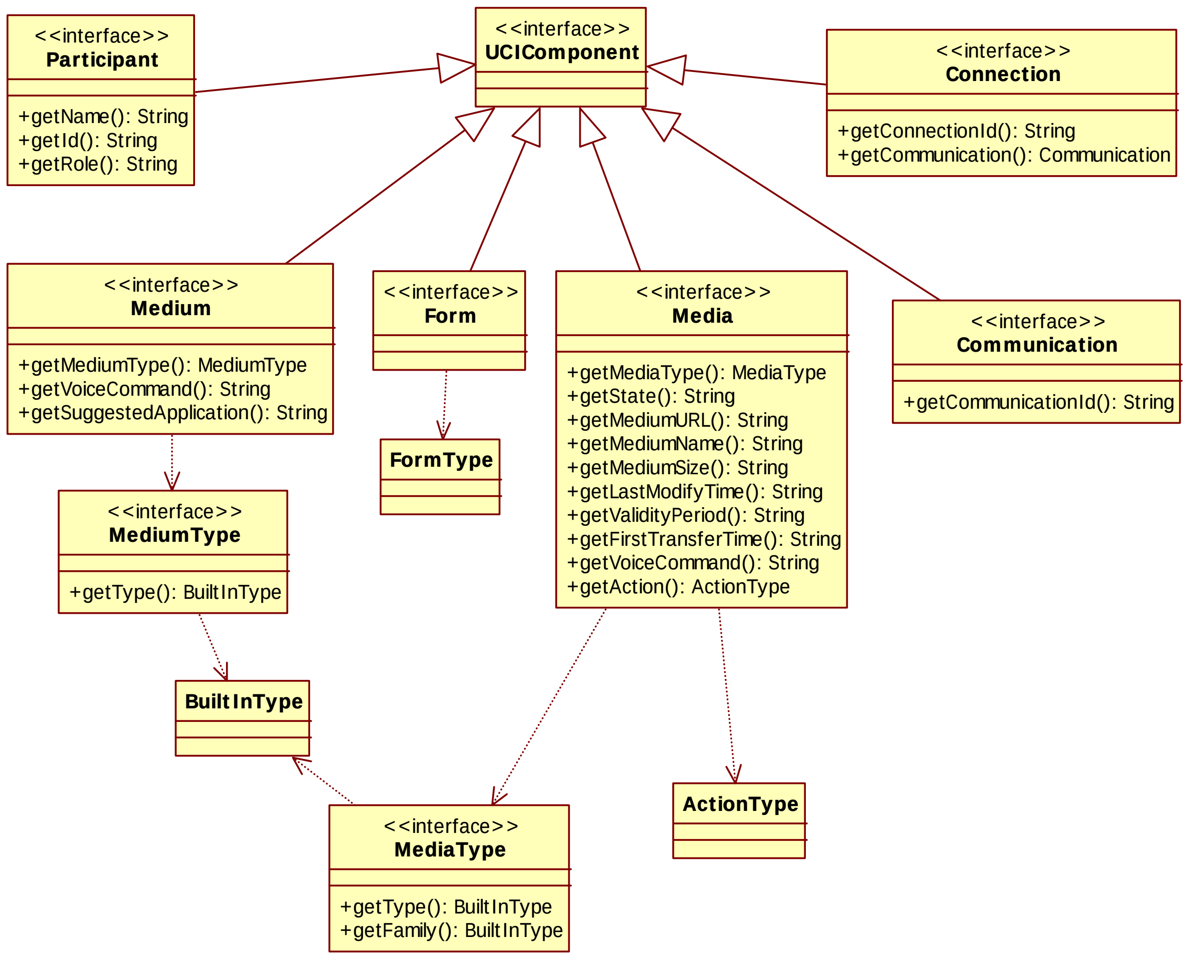


Figure 52 UCI.COMPONENT Package Class Diagram

10.4.9 REPOSITORY PACKAGE CLASS DIAGRAM

The class diagram below illustrates the relationships between the classes in the UCI.REPOSITORY Package.

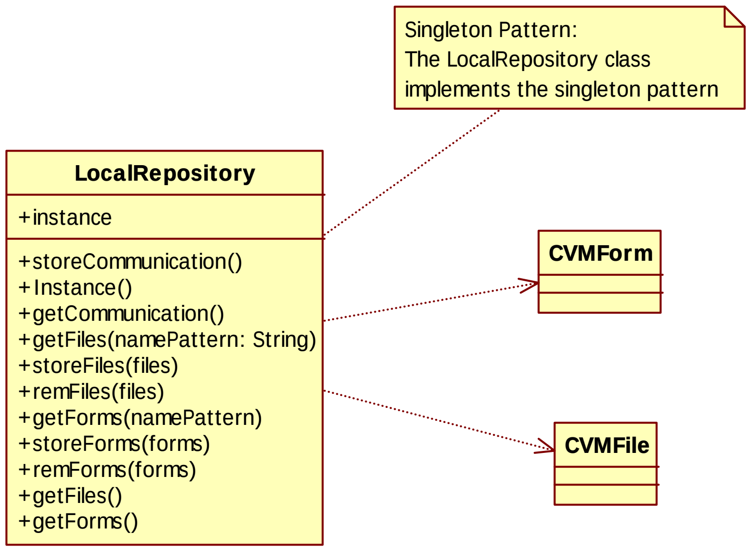


Figure 53 REPOSITORY Package Class Diagram

10.5 Appendix E - Class Interfaces

See attached API document for the class interfaces.

10.6 Appendix F – Test Driver

The following is the code for the UCI Subsystem Test Driver:

```
package edu.fiu.cis.cvm.uci;

import java.lang.reflect.Method;
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.LinkedBlockingQueue;
import java.util.concurrent.TimeUnit;

import edu.fiu.cis.cvm.uci.component.ActionType;
import edu.fiu.cis.cvm.uci.component.BuiltInType;
import edu.fiu.cis.cvm.uci.component.Communication;
import edu.fiu.cis.cvm.uci.component.Connection;
import edu.fiu.cis.cvm.uci.component.Media;
import edu.fiu.cis.cvm.uci.component.Medium;
import edu.fiu.cis.cvm.uci.component.MediumType;
import edu.fiu.cis.cvm.uci.component.Participant;
import edu.fiu.cis.cvm.uci.events.UCIEvent;
import edu.fiu.cis.cvm.uci.impl.component.CommunicationImpl;
import edu.fiu.cis.cvm.uci.impl.component.LiveAudioVideo;
import edu.fiu.cis.cvm.uci.impl.component.MediaImpl;
import edu.fiu.cis.cvm.uci.impl.component.MediumImpl;
import edu.fiu.cis.cvm.uci.impl.component.MediumTypeImpl;
import edu.fiu.cis.cvm.uci.impl.component.ParticipantImpl;
import edu.fiu.cis.cvm.uci.impl.component.TextFile;
import edu.fiu.cis.cvm.uci.impl.component.TextMessage;

/**
 * This is the Test Driver for the UCI Subsystem
 * date: 12/01/2008
 * author: Team 1
 */
public class UCITestDriver {

        public static void main(String[] args) throws Exception {

                Method methods[] = UCITestDriver.class.getDeclaredMethods();

                UCITestDriver driver = new UCITestDriver();

                for (Method method : methods) {
                        if (method.getName().startsWith("test")) {
                                method.invoke(driver);
                        }

                }

        }

        /**
         * Test Create Communication Sunny Day
 */
public void testCVMGUI_COM_001_UCI_001() {

	System.out.println("CVMGUI_COM_001_UCI_001");

	BlockingQueue<UCIEvent> queue = new LinkedBlockingQueue<UCIEvent>();

	try {

		Participant p1 = new ParticipantImpl("1", "Peter", "Faculty");
		Participant p2 = new ParticipantImpl("2", "John", "Student");

		UCI uci = UCIFactory.createUCI(queue,"1","Peter","Faculty");

		Communication communication = uci.createCommunication();
		Connection connection = uci.createConnection(communication);

		uci.addParticipant(connection, p1);
		uci.addParticipant(connection, p2);

		Console.readString("");

	} catch (Exception e) {
		e.printStackTrace();

	}

}

/**
 * Test Create Communication Sunny Day
 */
public void testCVMGUI_COM_001_UCI_002() {

	System.out.println("CVMGUI_COM_001_UCI_002");

	BlockingQueue<UCIEvent> queue = new LinkedBlockingQueue<UCIEvent>();

	try {

		Participant p1 = new ParticipantImpl("1", "George", "Surgeon");
		Participant p2 = new ParticipantImpl("2", "Ana",
					"Referring Phisician");

		UCI uci = UCIFactory.createUCI(queue,"1","George","Surgeon");

		Communication communication = uci.createCommunication();
		Connection connection = uci.createConnection(communication);

		uci.addParticipant(connection, p1);
		uci.addParticipant(connection, p2);

		Console.readString("");
```

```
        } catch (Exception e) {
                e.printStackTrace();

        }

}

/**
 * Test Load Communication Sunny Day
 */
public void testCVMGUI_COM_002_UCI_001() {

        System.out.println("CVMGUI_COM_002_UCI_001");

        BlockingQueue<UCIEvent> queue = new LinkedBlockingQueue<UCIEvent>();

        try {

                Communication communication = new CommunicationImpl("good");

                UCI uci = UCIFactory.createUCI(queue,"1","George","Surgeon");

                uci.loadCommunication(communication);

                for (int i = 0; i < 3; i++) {

                        System.out.println(queue.poll(1, TimeUnit.SECONDS));

                }

                Console.readString("");

        } catch (Exception e) {

                e.printStackTrace();

        }

}

/**
 * Test Load Communication Sunny Day
 */
public void testCVMGUI_COM_002_UCI_002() {

        System.out.println("CVMGUI_COM_002_UCI_002");

        BlockingQueue<UCIEvent> queue = new LinkedBlockingQueue<UCIEvent>();

        try {

                Communication communication = new CommunicationImpl("good2");

                UCI uci = UCIFactory.createUCI(queue,"1","Peter","Faculty");
```

```
			uci.loadCommunication(communication);

			for (int i = 0; i < 3; i++) {

				System.out.println(queue.poll(2, TimeUnit.SECONDS));

			}

			Console.readString("");

		} catch (Exception e) {

			e.printStackTrace();

		}

	}

	/**
	 * Test Load Communication Rainy Day
	 */
	public void testCVMGUI_COM_002_UCI_003() {

		System.out.println("CVMGUI_COM_002_UCI_003");

		BlockingQueue<UCIEvent> queue = new LinkedBlockingQueue<UCIEvent>();

		try {

			Communication communication = new CommunicationImpl("corrupt");

			UCI uci = UCIFactory.createUCI(queue,"1","Peter","Faculty");

			uci.loadCommunication(communication);

			Console.readString("");

		} catch (Exception e) {

			System.out.println(e.getClass() + ":"  + e.getMessage());

		}

	}

	/**
	 * Test Save Communication Sunny Day
	 */
	public void testCVMGUI_COM_005_UCI_001() {

		System.out.println("CVMGUI_COM_005_UCI_001");

		BlockingQueue<UCIEvent> queue = new LinkedBlockingQueue<UCIEvent>();
```

```
        try {

            Participant p1 = new ParticipantImpl("1", "Peter", "Faculty");
            Participant p2 = new ParticipantImpl("2", "John", "Student");

            UCI uci = UCIFactory.createUCI(queue,"1","Peter","Faculty");

            Communication communication = uci.createCommunication();
            Connection connection = uci.createConnection(communication);

            uci.addParticipant(connection, p1);
            uci.addParticipant(connection, p2);

            uci.storeCommunication(communication);

            Console.readString("");

        } catch (Exception e) {

            System.out.println(e.getClass() + ":"  + e.getMessage());

        }

    }


    /**
     * Test Save Communication Sunny Day
     */
    public void testCVMGUI_COM_005_UCI_002() {

        System.out.println("CVMGUI_COM_005_UCI_002");

        BlockingQueue<UCIEvent> queue = new LinkedBlockingQueue<UCIEvent>();

        try {

            Participant p1 = new ParticipantImpl("1", "Peter", "Faculty");
            Participant p2 = new ParticipantImpl("2", "John", "Student");
            Participant p3 = new ParticipantImpl("3", "Ana", "Student");

            UCI uci = UCIFactory.createUCI(queue,"1","Peter","Faculty");

            Communication communication = uci.createCommunication();
            Connection connection = uci.createConnection(communication);

            uci.addParticipant(connection, p1);
            uci.addParticipant(connection, p2);
            uci.addParticipant(connection, p3);

            uci.storeCommunication(communication);

            Console.readString("");
```

```
		} catch (Exception e) {

			System.out.println(e.getClass() + ":"  + e.getMessage());

		}

	}

	/**
	 * Test Add Participant Sunny Day
	 */
	public void testCVMGUI_COM_006_UCI_001() {

		System.out.println("CVMGUI_COM_006_UCI_001");

		BlockingQueue<UCIEvent> queue = new LinkedBlockingQueue<UCIEvent>();

		try {

			Participant p1 = new ParticipantImpl("1", "Peter", "Faculty");
			Participant p2 = new ParticipantImpl("2", "John", "Student");
			Participant p3 = new ParticipantImpl("3", "William", "Student");

			UCI uci = UCIFactory.createUCI(queue,"1","Peter","Faculty");

			Communication communication = uci.createCommunication();
			Connection connection = uci.createConnection(communication);

			uci.addParticipant(connection, p1);
			uci.addParticipant(connection, p2);
			uci.addParticipant(connection, p3);

			Console.readString("");

		} catch (Exception e) {

			System.out.println(e.getClass() + ":"  + e.getMessage());

		}

	}

	/**
	 * Test Add Participant Sunny Day
	 */
	public void testCVMGUI_COM_006_UCI_002() {

		System.out.println("CVMGUI_COM_006_UCI_002");

		BlockingQueue<UCIEvent> queue = new LinkedBlockingQueue<UCIEvent>();

		try {
```

```
                Participant p1 = new ParticipantImpl("1", "Peter", "Faculty");
                Participant p2 = new ParticipantImpl("2", "John", "Student");
                Participant p3 = new ParticipantImpl("3", "William", "Student");
                Participant p4 = new ParticipantImpl("4", "Ana", "Student");

                UCI uci = UCIFactory.createUCI(queue,"1","Peter","Faculty");

                Communication communication = uci.createCommunication();
                Connection connection = uci.createConnection(communication);

                uci.addParticipant(connection, p1);
                uci.addParticipant(connection, p2);
                uci.addParticipant(connection, p3);
                uci.addParticipant(connection, p4);

                Console.readString("");

        } catch (Exception e) {

                e.printStackTrace();

        }

}

/**
 * Test Add Participant Rainy Day (Duplicate Participant)
 */
public void testCVMGUI_COM_006_UCI_003() {

        System.out.println("CVMGUI_COM_006_UCI_003");

        BlockingQueue<UCIEvent> queue = new LinkedBlockingQueue<UCIEvent>();

        try {

                Participant p1 = new ParticipantImpl("1", "Peter", "Faculty");
                Participant p2 = new ParticipantImpl("2", "John", "Student");

                UCI uci = UCIFactory.createUCI(queue,"1","Peter","Faculty");

                Communication communication = uci.createCommunication();
                Connection connection = uci.createConnection(communication);

                uci.addParticipant(connection, p1);
                uci.addParticipant(connection, p2);
                uci.addParticipant(connection, p2);

                Console.readString("");

        } catch (Exception e) {

                e.printStackTrace();
```

```
		}

	}

	/**
	 * Test Enable Live Audio/Video Medium Sunny Day
	 */
	public void testCVMGUI_MED_002_UCI_001() {

		System.out.println("CVMGUI_MED_002_UCI_001");

		BlockingQueue<UCIEvent> queue = new LinkedBlockingQueue<UCIEvent>();

		try {

			Participant p1 = new ParticipantImpl("1", "Peter", "Faculty");
			Participant p2 = new ParticipantImpl("2", "John", "Student");

			UCI uci = UCIFactory.createUCI(queue,"1","Peter","Faculty");

			Communication communication = uci.createCommunication();
			Connection connection = uci.createConnection(communication);

			uci.addParticipant(connection, p1);
			uci.addParticipant(connection, p2);

			MediumType type = new MediumTypeImpl(BuiltInType.LIVE_AUDIO_VIDEO);

			Medium medium = new MediumImpl(type, "", "");

			uci.addMedium(connection, medium);

			Console.readString("");

		} catch (Exception e) {

			e.printStackTrace();

		}

	}

	/**
	 * Test Enable Live Audio/Video Medium Sunny Day
	 */
	public void testCVMGUI_MED_002_UCI_002() {

		System.out.println("CVMGUI_MED_002_UCI_002");

		BlockingQueue<UCIEvent> queue = new LinkedBlockingQueue<UCIEvent>();

		try {

			Participant p1 = new ParticipantImpl("1", "Peter", "Faculty");
			Participant p2 = new ParticipantImpl("2", "Ana", "Student");

			UCI uci = UCIFactory.createUCI(queue,"1","Peter","Faculty");

			Communication communication = uci.createCommunication();
			Connection connection = uci.createConnection(communication);

			uci.addParticipant(connection, p1);
			uci.addParticipant(connection, p2);

			MediumType type = new MediumTypeImpl(BuiltInType.LIVE_AUDIO_VIDEO);

			Medium medium = new MediumImpl(type, "", "");

			uci.addMedium(connection, medium);

			Console.readString("");

		} catch (Exception e) {

			e.printStackTrace();

		}

	}

	/**
	 * Test Enable Live Audio/Video Medium Rainy Day (Already Enabled)
	 */
	public void testCVMGUI_MED_002_UCI_003() {

		System.out.println("CVMGUI_MED_002_UCI_003");

		BlockingQueue<UCIEvent> queue = new LinkedBlockingQueue<UCIEvent>();

		try {

			Participant p1 = new ParticipantImpl("1", "Peter", "Faculty");
			Participant p2 = new ParticipantImpl("2", "Ana", "Student");

			UCI uci = UCIFactory.createUCI(queue,"1","Peter","Faculty");

			Communication communication = uci.createCommunication();
			Connection connection = uci.createConnection(communication);

			uci.addParticipant(connection, p1);
			uci.addParticipant(connection, p2);

			MediumType type = new MediumTypeImpl(BuiltInType.LIVE_AUDIO_VIDEO);

			Medium medium = new MediumImpl(type, "", "");

			uci.addMedium(connection, medium);
			uci.addMedium(connection, medium);
```

```
                Console.readString("");

        } catch (Exception e) {

                e.printStackTrace();

        }

}

/**
 * Test Disable Live Audio/Video Medium Sunny Day
 */
public void testCVMGUI_MED_003_UCI_001() {

        System.out.println("CVMGUI_MED_003_UCI_001");

        BlockingQueue<UCIEvent> queue = new LinkedBlockingQueue<UCIEvent>();

        try {

                Participant p1 = new ParticipantImpl("1", "Peter", "Faculty");
                Participant p2 = new ParticipantImpl("2", "John", "Student");

                UCI uci = UCIFactory.createUCI(queue,"1","Peter","Faculty");

                Communication communication = uci.createCommunication();
                Connection connection = uci.createConnection(communication);

                uci.addParticipant(connection, p1);
                uci.addParticipant(connection, p2);

                MediumType type = new MediumTypeImpl(BuiltInType.LIVE_AUDIO_VIDEO);

                Medium medium = new MediumImpl(type, "", "");

                uci.addMedium(connection, medium);

                uci.removeMedium(connection, medium);

                Console.readString("");

        } catch (Exception e) {

                e.printStackTrace();

        }

}

/**
 * Test Disable Live Audio/Video Medium Sunny Day
 */
```

```
public void testCVMGUI_MED_003_UCI_002() {

        System.out.println("CVMGUI_MED_003_UCI_002");

        BlockingQueue<UCIEvent> queue = new LinkedBlockingQueue<UCIEvent>();

        try {

                Participant p1 = new ParticipantImpl("1", "Peter", "Faculty");
                Participant p2 = new ParticipantImpl("2", "John", "Student");
                Participant p3 = new ParticipantImpl("2", "Ana", "Student");

                UCI uci = UCIFactory.createUCI(queue,"1","Peter","Faculty");

                Communication communication = uci.createCommunication();
                Connection connection = uci.createConnection(communication);

                uci.addParticipant(connection, p1);
                uci.addParticipant(connection, p2);
                uci.addParticipant(connection, p3);

                MediumType type = new MediumTypeImpl(BuiltInType.LIVE_AUDIO_VIDEO);

                Medium medium = new MediumImpl(type, "", "");

                uci.addMedium(connection, medium);

                uci.removeMedium(connection, medium);

                Console.readString("");

        } catch (Exception e) {

                e.printStackTrace();

        }

}

/**
 * Test Disable Live Audio/Video Medium Rainy Day (Already Disabled)
 */
public void testCVMGUI_MED_003_UCI_003() {

        System.out.println("CVMGUI_MED_003_UCI_003");

        BlockingQueue<UCIEvent> queue = new LinkedBlockingQueue<UCIEvent>();

        try {

                Participant p1 = new ParticipantImpl("1", "Peter", "Faculty");
                Participant p2 = new ParticipantImpl("2", "John", "Student");

                UCI uci = UCIFactory.createUCI(queue,"1","Peter","Faculty");
                Communication communication = uci.createCommunication();
                Connection connection = uci.createConnection(communication);

                uci.addParticipant(connection, p1);
                uci.addParticipant(connection, p2);

                MediumType type = new MediumTypeImpl(BuiltInType.LIVE_AUDIO_VIDEO);

                Medium medium = new MediumImpl(type, "", "");

                uci.addMedium(connection, medium);

                uci.removeMedium(connection, medium);
                uci.removeMedium(connection, medium);


                Console.readString("");

        } catch (Exception e) {

                e.printStackTrace();

        }

}

/**
 * Test Share File Sunny Day (Chat)
 */
public void testCVMGUI_MED_004_UCI_001() {

        System.out.println("CVMGUI_MED_004_UCI_001");

        BlockingQueue<UCIEvent> queue = new LinkedBlockingQueue<UCIEvent>();

        try {

                Participant p1 = new ParticipantImpl("1", "Peter", "Faculty");
                Participant p2 = new ParticipantImpl("2", "John", "Student");

                UCI uci = UCIFactory.createUCI(queue,"1","Peter","Faculty");

                Communication communication = uci.createCommunication();
                Connection connection = uci.createConnection(communication);

                uci.addParticipant(connection, p1);
                uci.addParticipant(connection, p2);

                Media media = new MediaImpl(ActionType.SEND, "", "",
                                new TextFile(), BuiltInType.TEXT_FILE.value(),
                                "", "", "", "", "", "");

                uci.shareMedia(connection, media);
                Console.readString("");

        } catch (Exception e) {

                e.printStackTrace();

        }

}

/**
 * Test Share File Sunny Day (Chat)
 */
public void testCVMGUI_MED_004_UCI_002() {

        System.out.println("CVMGUI_MED_004_UCI_002");

        BlockingQueue<UCIEvent> queue = new LinkedBlockingQueue<UCIEvent>();

        try {

                Participant p1 = new ParticipantImpl("1", "Peter", "Faculty");
                Participant p2 = new ParticipantImpl("2", "John", "Student");
                Participant p3 = new ParticipantImpl("2", "Ana", "Student");

                UCI uci = UCIFactory.createUCI(queue,"1","Peter","Faculty");

                Communication communication = uci.createCommunication();
                Connection connection = uci.createConnection(communication);

                uci.addParticipant(connection, p1);
                uci.addParticipant(connection, p2);
                uci.addParticipant(connection, p3);

                Media media = new MediaImpl(ActionType.SEND, "", "",
                                new TextFile(), BuiltInType.TEXT_FILE.value(),
                                 "", "", "", "", "", "");

                uci.shareMedia(connection, media);

                Console.readString("");

        } catch (Exception e) {

                e.printStackTrace();

        }

}

/**
 * Test Start Live Audio/Video Sunny Day
 */
```

```
public void testCVMGUI_MED_005_UCI_001() {

        System.out.println("CVMGUI_MED_005_UCI_001");

        BlockingQueue<UCIEvent> queue = new LinkedBlockingQueue<UCIEvent>();

        try {

                Participant p1 = new ParticipantImpl("1", "Peter", "Faculty");
                Participant p2 = new ParticipantImpl("2", "John", "Student");

                UCI uci = UCIFactory.createUCI(queue,"1","Peter","Faculty");

                Communication communication = uci.createCommunication();
                Connection connection = uci.createConnection(communication);

                uci.addParticipant(connection, p1);
                uci.addParticipant(connection, p2);

                Media media = new MediaImpl(ActionType.START, "", "",
                                new LiveAudioVideo(), BuiltInType.LIVE_AUDIO_VIDEO.value(),
                                "", "", "", "", "", "");

                uci.startMedia(connection, media);

                Console.readString("");

        } catch (Exception e) {

                e.printStackTrace();

        }

}

/**
 * Test Stop Live Audio/Video Sunny Day
 */
public void testCVMGUI_MED_005_UCI_002() {

        System.out.println("CVMGUI_MED_005_UCI_002");

        BlockingQueue<UCIEvent> queue = new LinkedBlockingQueue<UCIEvent>();

        try {

                Participant p1 = new ParticipantImpl("1", "Peter", "Faculty");
                Participant p2 = new ParticipantImpl("2", "John", "Student");

                UCI uci = UCIFactory.createUCI(queue,"1","Peter","Faculty");

                Communication communication = uci.createCommunication();
                Connection connection = uci.createConnection(communication);
```

```
                        uci.addParticipant(connection, p1);
                        uci.addParticipant(connection, p2);

                        Media media = new MediaImpl(ActionType.START, "", "",
                                        new LiveAudioVideo(), BuiltInType.LIVE_AUDIO_VIDEO.value(),
                                        "", "", "", "", "", "");

                        uci.startMedia(connection, media);

                        uci.stopMedia(connection, media);

                        Console.readString("");

                } catch (Exception e) {

                        e.printStackTrace();

                }

        }

        /**
         * Test Start Live Audio/Video Rainy Day (Already Started)
         */
        public void testCVMGUI_MED_005_UCI_003() {

                System.out.println("CVMGUI_MED_005_UCI_003");

                BlockingQueue<UCIEvent> queue = new LinkedBlockingQueue<UCIEvent>();

                try {

                        Participant p1 = new ParticipantImpl("1", "Peter", "Faculty");
                        Participant p2 = new ParticipantImpl("2", "John", "Student");

                        UCI uci = UCIFactory.createUCI(queue,"1","Peter","Faculty");

                        Communication communication = uci.createCommunication();
                        Connection connection = uci.createConnection(communication);

                        uci.addParticipant(connection, p1);
                        uci.addParticipant(connection, p2);

                        Media media = new MediaImpl(ActionType.START, "", "",
                                        new LiveAudioVideo(), BuiltInType.LIVE_AUDIO_VIDEO.value(),
                                        "", "", "", "", "", "");

                        uci.startMedia(connection, media);
                        uci.startMedia(connection, media);

                        Console.readString("");

                } catch (Exception e) {
```

```
                e.printStackTrace();

            }

        }

        /**
         * Test Send Chat Message Sunny Day
         */
        public void testCVMGUI_MED_006_UCI_001() {

            System.out.println("CVMGUI_MED_006_UCI_001");

            BlockingQueue<UCIEvent> queue = new LinkedBlockingQueue<UCIEvent>();

            try {

                Participant p1 = new ParticipantImpl("1", "Peter", "Faculty");
                Participant p2 = new ParticipantImpl("2", "John", "Student");

                UCI uci = UCIFactory.createUCI(queue,"1","Peter","Faculty");

                Communication communication = uci.createCommunication();
                Connection connection = uci.createConnection(communication);

                uci.addParticipant(connection, p1);
                uci.addParticipant(connection, p2);

                Media media = new MediaImpl(ActionType.SEND, "", "",
                        new TextMessage(), BuiltInType.TEXT_MSG.value(),
                        "", "", "", "", "", "");

                uci.shareMedia(connection, media);

                Console.readString("");

            } catch (Exception e) {

                e.printStackTrace();

            }

        }

        /**
         * Test Send Chat Message Sunny Day
         */
        public void testCVMGUI_MED_006_UCI_002() {

            System.out.println("CVMGUI_MED_006_UCI_002");

            BlockingQueue<UCIEvent> queue = new LinkedBlockingQueue<UCIEvent>();

            try {
                    Participant p1 = new ParticipantImpl("1", "Peter", "Faculty");
                    Participant p2 = new ParticipantImpl("2", "John", "Student");
                    Participant p3 = new ParticipantImpl("3", "Ana", "Student");

                    UCI uci = UCIFactory.createUCI(queue,"1","Peter","Faculty");

                    Communication communication = uci.createCommunication();
                    Connection connection = uci.createConnection(communication);

                    uci.addParticipant(connection, p1);
                    uci.addParticipant(connection, p2);
                    uci.addParticipant(connection, p3);

                    Media media = new MediaImpl(ActionType.SEND, "", "",
                                        new TextMessage(), BuiltInType.TEXT_MSG.value(),
                                        "", "", "", "", "", "");

                    uci.shareMedia(connection, media);

                    Console.readString("");

            } catch (Exception e) {

                    e.printStackTrace();

            }

      }


}
```

10.7 Appendix G – Diary of Meetings

Date: 09-04-2008

Start Time: 7:00 PM

End Time: 8:30 PM

Location: ECS-234

Minute Taker: Barbara

Members in Attendance: Barbara, Jorge, Ricardo, David, Hong, Nathanael

Members late to the Meeting: None

***Agenda*:**

- Define a preliminary conceptual model for the CVM GUI

- Specify the use cases for the CVM GUI

- Assign use cases to each team member

***Discussion*:**

- The discussion focused on getting to understand the concepts managed for the CVM and identifying some scenarios.

- We defined a preliminary list of use cases for the CVM GUI.

- Each team member was assigned to a subset of the system use cases.

- Since the requirements are not still very clear, we concluded that it is necessary to discuss them further with some application domain expert.

Additionally we consider that it would be useful to have some screenshots from the prototype in order to understand better its functionalities.

- We identified a risk for the project: none of the members of the team is skilled in the development of desktop GUIs.

Assignments for Next Meeting:

- Nate volunteered for having a meeting with Prof. Clarke or Andrew Allen to discuss further the requirements.

- Each team member agreed to write their assigned use cases.

Date: 09-11-2008

Start Time: 6:30 PM

End Time: 8:15 PM

Location: ECS-234

Minute Taker: Barbara

Members in Attendance: Barbara, Jorge, Eddie, Ricardo, David, Hong, Nathanael

Members late to the Meeting: None

Agenda:

- Check and briefly discuss the list of use cases.

- Assign use cases to each team member.

- Make a list of all our open questions.

Discussion:

- During the meeting we went through each of the use cases already identifed and briefly discussed them.

- Some new use cases were added to the list, including misuse and security ones.

- A subset of the use cases in the resulting list was assigned to each team member. The assignments are published in the project website.

- The following open questions were identified:

  * Are we going to treat file containers as devices? For example, an user adds a file cabinet to a communication and then is allowed to send files through it?

  * Where does the list of devices come from?

  * Where does all the configuration of the devices come from?

  * where is the user information stored/loaded from? Login, password...

- We are going to use a page in the project website to keep track of all open questions.

- Each use case is going to be written in a separate file using the template available in the project website. The name of the file will match the use case name.

- The use case drafts produced this week should be published on the project website.

Assignments for Next Meeting:

- Each team member agreed to write a first draft their assigned use cases using the template published in the website.

- Get clarification for all open questions.

Date: 09-18-2008

Start Time: 6:30 PM

End Time: 8:00 PM

Location: ECS-212, ECS-234

Minute Taker: Barbara

Members in Attendance: Barbara, Jorge, Eddie, Ricardo, David, Hong, Nathanael

Members late to the Meeting: None

Agenda:

- Define the tasks for the following week.

- Obtain clarification from our open questions.

Discussion:

- We should start working immediately on a GUI prototype.

- The GUI must be implemented in Java using the Eclipse Framework.

- It is important to divide our team in sub teams specific for each aspect of the SRD.

- We are dividing the team in the following sub teams:

    * GUI: Jorge, Ricardo, David

    * UCI: Nate, Hong

    * Documentation: Barbara, Eddie

- The GUI is our priority for next week.

- We have to get the latest version of the xsd document for X-CML.

- Jorge will get a SVN repository working by tonight and send the repository information to the rest of the group.

- We are going to be using Subclipse for integrating SVN file management with eclipse.

Assignments for Next Meeting:

All:

- Upload all use cases to the website

GIU & UCI Teams:

- Set up SVN repository for configuration management.

- Implement non-functional prototype.

Documentation Team:

- Build first draft of the project plan.

- Review the Use Cases checking for consistency.

- Create use case diagrams.

- Start working on scenarios, object diagrams and sequence diagrams.

Date: 09-25-2008

Start Time: 6:30 PM

End Time: 8:00 PM

Location: ECS-212, ECS-234

Minute Taker: Barbara

Members in Attendance: Barbara, Jorge, Eddie, Ricardo, David, Hong, Nathanael

Members late to the Meeting: None

Agenda:

- Define responsibility of each team member regarding the Use Cases.

- Get clarification on the functionalities of the CVM GUI related with managing forms.

- Define the screens that will be implemented on the non-functional prototype.

- Agree on the Misuse Case and Security Use Case Templates.

Discussion:

- The following use cases will be removed from the use case model: Remove Participant, Edit Contact, Start Mediator, Stop Mediator, Add Device, Remove Device.

- The following use cases will be added: Share Specific Form, Share Generic Form, Build Generic Form, Update Communication Schema (Synthesis Engine).

- The following use cases will be rewritten to support multicasting: Add Live Stream Media, Remove Live Stream Media, Mute Live Audio Stream.

- The use cases should be updated to reflect the conceptual difference between communication and connection.

- We are considering only 2 actors for our use cases: User & Synthesis Engine.

- We are using CVMGUI_COM_001 as a template for the use cases.

- We are using CVMGUI_MIS_001 as a template for the misuse cases.

- We are using CVMGUI_SEC_001 as a template for security use cases.

- Jorge is collaborating with the Documentation Team to help in the edition of the Use Cases.

- David is collaborating with the Documentation Team for building the project cost estimate.

Assignments for Next Meeting:

All:

- Update Use Cases by Sunday.

- Everything should be finished by Sunday October 5th.

GUI Team:

- Screenshots for everything on the use cases.

UCI Team:

- Build example xmls for scenarios.

Documentation Team:

- Build a list of the required xmls for the analysis model.

- Put the Use Cases together.

- Write & Review the Introduction.

- Update and Finalize the Analysis Model.

- Update Project Plan.

Date: 10-01-2008

Start Time: 8:30 PM

End Time: 9:30 PM

Location: ECS-234

Minute Taker: Barbara

Members in Attendance: Barbara, Jorge, Eddie, Ricardo, David, Nathanael

Members late to the Meeting: None

Agenda:

- Communicate the status of sub team

- Divide the presentation slides among team members

- Review work products and determine all missing parts of the SRD.

Discussion:

- On Monday October 9th we will have a meeting for rehearsal of the presentation.

- Nate and Barbara will meet with Yingbo to get clarification on the X-CML specification.

- Hong and Eddie will be responsible for building the presentation slides. They should have this ready by Sunday.

- Nathanael will print and bind the SRD.

- Barbara and Jorge will review the updated use cases.

- David will provide the cost estimate for the project, he has almost finished it.

- The GUI team should build a document with all the screenshots and update the use cases to add references to them.

Assignments for Next Meeting:

- Hong and Eddie will have the presentation slides ready.

- Nathanael will build the X-CML schemas for each of the scenarios.

- Barbara will prepare the use case diagrams and update the project schedule for the next phases of the project.

- Eddie should have finished the presentation slides.

- The GUI team should have finished the screenshots.

Date: 10-06-2008

Start Time: 9:00 PM

End Time: 10:00 PM

Location: ECS-234

Minute Taker: Barbara

Members in Attendance: Barbara, Jorge, Eddie, Ricardo, David, Hong, Nathanael

Members late to the Meeting: Barbara, Hong

Agenda:

- Review and rehearse the presentation.

- Determine missing parts of the SRD.

Discussion:

- The presentation slides were reviewed.

- The slides need to be summarized. Each presenter will be responsible for updating their slides.

- Nathanael will prepare the approvals section of the SRD.

- Jorge will build the project glossary.

- Everyone should have their SRD updates ready by tomorrow.

Assignments for Next Meeting:

- Each member should submit their assigned SRD and presentation slide updates by Tuesday.

- Barbara will finish assembling the SRD.

- Nathanael will print and bind the SRD.

Date: 10-09-2008

Start Time: 6:30 PM

End Time: 8:30 PM

Location: ECS-234

Minute Taker: David

Members in Attendance: Barbara, Jorge, Eddie, Ricardo, David, Hong, Nathanael

Members late to the Meeting: None

Agenda:

- Start discussion of the second project phase (Design Documents).

- Go over the DD template and clarify any misunderstandings.

- Assign tasks for the next meeting.

Discussion:

- We passed out the temple for the Design Document phase to all team members. Went through each section of the template and projected future tasks to be accomplished during this phase.

- A task of summarizing Functional and Nonfunctional Requirements was given to Eddie.

- Following the discussion of the DD template the team began to consider the Architectural and Design patter for our project. We all agreed on Repository and MVC architectural pattern.

- The team then presented a design plan for our project. Nate explained different UCI controls and how they would interact with the GUI component. We are going to use two queues, one for incoming and the other is for outgoing events. GUI would have an event listener that checks the queue for incoming events until empty.

Assignments for Next Meeting:

- Eddie to summarize Functional and Nonfunctional Requirements.

- GUI team to optimize and clean up source code for the next meeting.

Date: 10-16-2008

Start Time: 6:30 PM

End Time: 8:30 PM

Location: ECS-234

Minute Taker: David

Members in Attendance: Barbara, Jorge, Eddie, Ricardo, David, Hong, Nathanael

Members late to the Meeting: None

Agenda:

- Continue discussion of the second project phase (Design Documents).

- Begin incorporating UCI and GUI components.

- Fully agree on Architectural Pattern.

Discussion:

- Nate has finalized U-CML schema, and it was then validated with the other group's members.

- We then spent the rest of the meeting discussing and clarifying UCI functionality. Nate opened up the CVM project and we went over each of the UCI class packages. UCI is mainly broken down into interfaces, events, and components, for each there are different types of objects and all work in unity to make our program functional. We then discussed the communicational aspect of the project, in short:

1. X-CML is retrieved from the Synthesis Engine.
2. UCI parses the code and populates needed objects.
3. The events along with components are then pushed into the queue and delivered to GUI.
4. GUI retrieves the objects in the queue and populates the interfaces.
5. The inverse of this process is valid for actions produced by the user via GUI, instead now the X-CML code is created and gets passed down to the Synthesis Engine.

- We then resolved Eddie's issue with summarizing Functional and Non-Functional Requirements.

Assignments for Next Meeting:

- Further populate UCI missing components and values.

- Completely envelop each other with knowledge of UCI and GUI interaction.

Date: 10-23-2008

Start Time: 6:30 PM

End Time: 9:00 PM

Location: ECS-234

Minute Taker: David

Members in Attendance: Barbara, Jorge, Eddie, Ricardo, David, Hong, Nathanael

Members late to the Meeting: None

Agenda:

- Start working on Design Document

- Architecture Pattern

- Additional Events for UCI and GUI Communications

Discussion:

- The team began working on the Design Document, by going over each of the subsections of the template provided by Prof. Clarke.

- Layered Architecture is the official pattern we are using for our system.

- We added additional Events to the UCI subsystem.

- We have to add new component packages to our project, which are Differentiating Engine, and Synthesis Engine.

Assignments for Next Meeting:

- Nate (UCI State Machine, Fix Event Trees)
- David (GUI State Machine, (SE) Synchronized Queue of Strings, Data In, Data Out, Control In, Control Out, Event, Document GUI Classes)
- Jorge (OCL)
- Barbara (Detailed Class Diagrams, Façade)
- Ricardo (Specification Repository Component, Document GUI Classes)

- Eddie (DE)
- Hong (Document UCI Classes)

Date: 10-30-2008

Start Time: 6:30 PM

End Time: 9:45 PM

Location: ECS-234

Minute Taker: David

Members in Attendance: Barbara, Jorge, Eddie, Ricardo, David, Hong, Nathanael

Members late to the Meeting: None

Agenda:

- Finish up with DD

- Work on a Sequence diagrams

- Discuss Control Class for GUI

Discussion:

- We discussed the necessary chapters to be done for DD
- The team then explained all the needed Events for projects, and how they would interact with the GUI controller
- The team then clarified the issues with State Machines in Professors Clarke office.

Assignments for Next Meeting:

- Introduction to the Overview (Paragraph "This section contains…" overview of document and Intro to the Introduction)(extend the Abstract, this is interesting…)(David)
- Functional Requirements (Eddie)
- Update Section (1.4) Definitions, Glossary (Hong)
- Chapter 2 (Barbara)
- Overview of the Document (Briefly describe all sections in 1.5)(Jorge)

- 3.4 Detailed Class Design (Hong, David)
- Minimal Class Diagram Section 3.1 (Jorge)
- Control Class for GUI (David, Ricardo, Jorge)
- State Diagram, JavaDoc (Nate)
- Repository(Ricardo)

Date: 11-06-2008

Start Time: 6:30 PM

End Time: 9:45 PM

Location: ECS-234

Minute Taker: David

Members in Attendance: Barbara, Jorge, Eddie, Ricardo, David, Hong, Nathanael

Members late to the Meeting: None

Agenda:

- Work on Sequence Diagrams

- Clear up State Machine Diagrams

- Finalize Design Document

- Begin Discussing Implementation Stage

Discussion:

- The teams started the meeting with Jorges queries about specifics concerning UCL
- We then created few Sequence Diagrams which would be used as the base cases.
- The meeting adjourned with assignment distributions

Assignments for Next Meeting:

- GUI State Machine, append to GUI StarUML (David)
- Functional Requirements (Eddie)

- Update Section (1.4) Definitions, Glossary (Hong)
- Sequence Diagrams (Barbara)
- Overview of the Document (Briefly describe all sections in 1.5)(Jorge)
- 3.4 Detailed Class Design (Hong, David)
- Minimal Class Diagram Section 3.1 (Jorge)
- State Diagram, JavaDoc (Nate)
- Repository(Ricardo)

Date: 11-13-2008

Start Time: 6:35 PM

End Time: 8:15 PM

Location: ECS-234 and ECS-212

Minute Taker: Jorge

Members in Attendance: Barbara, Jorge, Ricardo, David, Hong, Nathanael

Members late to the Meeting: None

Agenda:

- Divide and assign tasks for phase III
- Agree on how to limit the scope of the project

Discussion:

- The discussion focused on which functionality was key to system and which could be postponed.
- We analyzed the requirements for the phase III, which are mainly focused on testing.
- We agreed with the client on the functionality required for the system, hence postponing the implementation of use cases related to generic and specific forms.
- We divided the tasks for the next phase and established as one week the time to have the first prototype of the system working.

Assignments for Next Meeting:

- Ricardo and Jorge:Implementation of the GUI controller, GUI events and repository
- David: Implementation of the UCI X-CML schema differencing engine
- Barbara: Implementation of the command pattern for the elements of the X-CML schema.
- Nathanael: Implementation of the X-CML validation engine.
- Hong: Implementation of the Synthesis Engine driver.
- Hong and Eddie: start the specification of the test cases.

Date: 11-19-2008

Start Time: 7:50 PM

End Time: 9:30 PM

Location: ECS-234

Minute Taker: Jorge

Members in Attendance: Barbara, Jorge, Ricardo, David, Hong, Nathanael

Members late to the Meeting: None

Agenda:

- Discuss integration and test cases.

Discussion:

- The discussion focused on the integration of the two mayor subsystems of our application: GUI and UCI.
- We also identified which parts of the final document are pending.

Assignments for Next Meeting:

- Everyone is to continue working on getting their implementation to work and write the test cases.

Date: 11-26-2008

Start Time: 6:40 PM

End Time: 8:30 PM

Location: ECS-234

Minute Taker: Jorge

Members in Attendance: Barbara, Jorge, Ricardo, David, Hong, Nathanael

Members late to the Meeting: None

Agenda:

- Discuss integration and test cases.

Discussion:

- The discussion focused deciding concrete test cases.
- Some issues still need attention in terms of integration.

Assignments for Next Meeting:

- Everyone is to continue working on getting their implementation to

work and write the test cases.

Date: 12-02-2008

Start Time: 5:40 PM

End Time: 8:40 PM

Location: ECS-234

Minute Taker: Jorge

Members in Attendance: Barbara, Jorge, Ricardo, David, Nathanael

Members late to the Meeting: None

Agenda:

- Elaborate Presentation.
- Fix integration issues.

Discussion:

- We made, distributed and practiced the final presentation.

Final pending

- Integration issues were resolved.

Assignments for Next Meeting:

- Everyone is to continue working on writing their test cases and

resolve issues regarding their behavior.