CEN 5064 Software Design

**Rapid Realization of Communication Services System**

**Group #5 – Healthcare**

**Jean Rodriguez**

**Harika Chirumamilla**

**Denis Antoine**

**Marcelo R. Lopez, Jr.**

**Final Deliverable**

**April 14, 2009**

**Professor: Dr. Peter Clarke**

**Abstract**

Users have many options available today in the realm of online communication and collaboration; it’s difficult for the user to make a choice from the available alternatives. The considerable number of communication and collaborative applications available today makes it hard for the active consumer to define an optimal platform in terms of existing mediums. Each of these services provide unique features, such as chat, audio, video-conferencing, and file transferring, but not one offers them all at once. Furthermore, they have been created independently therefore no connections exist among them. Ultimately, End users will be able to set up a combination of communication models, simply by setting up the connections in the modeling environment.

The Rapid Realization of Communication Services System (RRComSSys) proposes a system that is a self-configuring, user-centric, communication service that brings together the available communication services, such as Skype, GoogleTalk, and others, chosen by the user. In addition, it provides access to all their capabilities, and it takes care of the creation of the connection protocols relieving the user of such burden. This document builds from the Software Requirements Document and Design Document. This document also focuses on the object design. The object design consists mainly of the minimal class diagram(s), sequence diagrams, and state machines.

In, development, the unified software development process (USDP), will be used. It is a component-based, use case driven, architecture centered, iterative and incremental developmental process that uses the Unified Modeling Language (UML) to represent models of the software. The USDP approach was complemented with the use of the Model Driven Software Development (MDSD) approach. Domain analysis, Meta modeling, model-driven generation, template languages, domain-driven framework design, and the principles for agile software development form the backbone of this approach. MDSD models have the exact meaning of program code in the sense that the bulk of the final implementation can be generated from them.

Table of Contents

[1 Introduction 5](#_Toc227492897)

[1.1 Purpose of system 5](#_Toc227492898)

[1.2 Scope of system 6](#_Toc227492899)

[1.3 Current System 6](#_Toc227492900)

[1.4 Analysis and Design Methodology used 6](#_Toc227492901)

[1.5 Definitions, Acronyms, and Abbreviations 8](#_Toc227492902)

[1.6 Overview of the document 9](#_Toc227492903)

[2 Project Plan 10](#_Toc227492904)

[2.1 Project organization – assignment of roles for the entire project. 10](#_Toc227492905)

[2.2 Hardware and Software Requirements 13](#_Toc227492906)

[3 Requirements elicitation and analysis 15](#_Toc227492907)

[3.1 Overview 15](#_Toc227492908)

[3.2 Functional Requirements 15](#_Toc227492909)

[3.3 Non functional requirements 16](#_Toc227492910)

[3.4 System Models 17](#_Toc227492911)

[3.4.1 Use case model (See Appendix B) 17](#_Toc227492912)

[3.4.2 Object model 17](#_Toc227492913)

[3.4.3 Dynamic model **Error! Bookmark not defined.**](#_Toc227492914)

[3.4.4 User Interfaces 21](#_Toc227492915)

[3.5 Validation of the analysis model 21](#_Toc227492916)

[3.5.1 Test Cases. 21](#_Toc227492918)

[3.5.2 Checklist 36](#_Toc227492919)

[3.5.3 Static execution. 37](#_Toc227492920)

[4 Proposed Software Architecture 40](#_Toc227492921)

[4.1 Overview 40](#_Toc227492922)

[4.2 Meta-model for the DSL 42](#_Toc227492923)

[4.3 UML profiles to represent the architecture 43](#_Toc227492924)

[4.4 Generative Architecture 45](#_Toc227492925)

[4.5 Subsystem Decomposition – describe each of the major subsystems 46](#_Toc227492926)

[4.6 Validation of the System Model 47](#_Toc227492927)

[4.6.1 Checklist 47](#_Toc227492928)

[4.6.2 Static execution 49](#_Toc227492929)

[5. Object Design 51](#_Toc227492930)

[5.1 Overview – minimal class diagrams for the subsystems 51](#_Toc227492931)

[5.2 Object Interaction 53](#_Toc227492932)

[5.2.1 Sequence Diagram 53](#_Toc227492936)

[5.2.2 State Machine 54](#_Toc227492937)

[5.3 Detailed Class Design 54](#_Toc227492938)

[5.4 Validation of the system model 56](#_Toc227492939)

[5.4.1 Checklist 56](#_Toc227492940)

[5.4.2 Static Execution 57](#_Toc227492941)

[6. Implementation 58](#_Toc227492942)

[6.1 Platform Specific Model (PSM) 58](#_Toc227492943)

[6.2 Validation of the Implementation 59](#_Toc227492944)

[7. Glossary 60](#_Toc227492945)

[8. Appendix 62](#_Toc227492946)

[8.1 Appendix A – Use case diagram 62](#_Toc227492947)

[8.2 Appendix B - Use Cases 65](#_Toc227492948)

[Use Case – Create a Text Only Connection 65](#_Toc227492949)

[Use Case – Create a Text Audio Connection 66](#_Toc227492950)

[Use Case – Create a Audio Video Connection 68](#_Toc227492951)

[Use Case – Send File 70](#_Toc227492952)

[Use Case – Receive File 71](#_Toc227492953)

[Use Case – Security File Transfer 73](#_Toc227492954)

[Use Case – Workflow with Alternate Connection 74](#_Toc227492955)

[Use Case – Workflow with Retry 77](#_Toc227492956)

[8.3 Appendix C – Object Diagram 79](#_Toc227492957)

[8.4 Appendix D: Sequence diagrams 83](#_Toc227492964)

[8.5 Appendix E –User Interfaces 91](#_Toc227492965)

[8.6 Appendix F – Detailed Class Diagram 97](#_Toc227492966)

[8.7 Appendix G – Class Interfaces 100](#_Toc227492969)

[8.8 Appendix H - Project schedule 105](#_Toc227492970)

[8.9 Appendix I – Diary of meeting and tasks. 106](#_Toc227492971)

# Introduction

This document constitutes the Final Deliverable for the Rapid Realization of Communication Services System (RRComSSys). RRComSSys is part of the Communication Virtual Machine (CVM) project developed at the School of Computer and Information Science at Florida International University.

This chapter covers the general description of the system including its purpose and its scope from the user’s point of view. The limitations of the current system are explained, also the analysis and design methodology used, as well as the definitions, acronyms and any abbreviations used in this paper. It concludes with an overview of the general structure of the document.

## Purpose of system

The RRComSSys has four phases: 1) The creation of the communication model by the user thru a visual environment based on a meta-model defined by the system, 2) the transformation of this model, a Graphical communication model (G-CML), into an XML communication model (X-CML), 3) the validation of the communication model for realization, and 4) the realization of the model by making the appropriate calls to the communication services.

The domain of our project is healthcare. (Other teams participating in this course cover other domains such as education, disaster management, financial and scientific collaboration. Refer to the course webpage for more information). This implies that our use cases and scenarios are based on situations that take place in a healthcare environment.

## Scope of system

The scope of the system is defined by the time a user starts the system and establishes communication with other users. An active session may include but is not limited to chat, audio, file transfer and video conference. The scope of the system will end when all users are disconnected from the system and therefore there is no more interaction. End users will be in the medical field so we can expect a higher level of technical skill even though the average user can operate the system.

## Current System

Currently in the healthcare field there is a variety of communication methods used between professionals. Mainly a telephone is used for verbal communication. Without a doubt email is the number one method used to send and receive files between users. In many cases the professionals would have to meet in person due to limitations of a phone conversation or a video conference. RRComSSys aims to become the number one communication system used by medical professionals. The user will have a single tool set which is flexible and will take the place of all of the communication methods currently available.

## Analysis and Design Methodology used

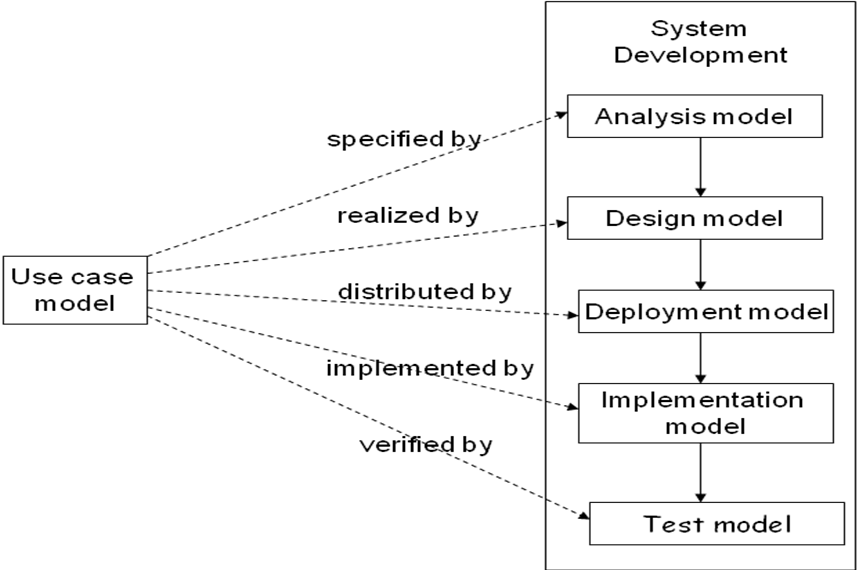
The software development model that has been used in the project is Unified Software Development Process (USDP). It is a component-based, use case driven, architecture centered, iterative and incremental developmental process that uses the Unified Modeling Language (UML) to represent models of the software system to be developed.

The main reason is because it provides traceability features, which is important as it provides means for mapping model artifacts among several stages of the project, and it is use case driven.

The iterative and incremental features help refine the final product as we get to know specific implementation platforms, namely Eclipse GMF and the Skype library. The use case approach for gathering the systems requirements was also suitable to collect the functional requirements in this project.

In addition, we eased the design of the system by using architectural and design patterns. The architectural patterns used are: Model View Controller, Pipe and Filter. We used the UML 2.0 notation for specifying the different artifacts of the system. The UML models used in the project are: uses case diagrams, class diagrams, sequence diagrams, UML profiles.

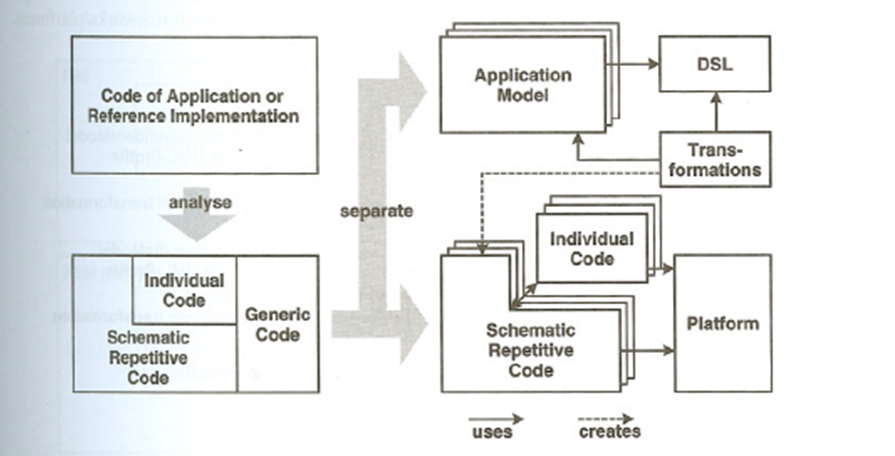
The following figure shows how the different phases of the USDP process are related.



**Figure 1.4.1** Unified Software Development Process

The USDP approach was complemented with the use of the Model Driven Software Development (MDSD) approach. Domain analysis, Meta-modeling, model-driven generation, template languages, domain-driven framework design, and the principles for agile software development form the backbone of this approach. MDSD models have the exact meaning of program code in the sense that the bulk of the final implementation can be generated from them. The main reason for using the MDSD approach was to increase the development speed. Since by automation run able code is generated from the formal models using one or more transformation steps. MDSD manages the complexity through abstraction since modeling languages enable programming or configuration on a more abstract level. Reusability was another big reason this approach was chosen, once all our architectures, transformations, and modeling languages have been defined they can be later used in a software product line for manufacturing diverse software systems.

The following figure shows how the different phases of the MDSD process are related.



**Figure 1.4.2 Model Driven Software Development**

The total approach is the combination of the unified software development process and the model driven software development process. Unified Software Development Process uses the Model Driven Software Development which results in an increase in development speed, better software quality, reusability, interoperability and the portability.

## Definitions, Acronyms, and Abbreviations

CVM: Communication Virtual Machine.

RRComSSys: Rapid Realization of Communication Services System.

VE: Visual Environment

MT: Model Transformer

STE: Schema Transformation Environment

NCB: Network Communication Broker.

SE: Synthesis Engine

CML: Communication Modeling Language

XML: eXtensible Markup Language.

G-CML: Graphical Communication Modeling Language

X-CML: XML-based Communication Modeling Language.

System: self configuration communication tool.

MDSD: Model-Driven Software Development

USDP: Unified Software Development Process

Project Stakeholder: Professor Peter Clarke

## Overview of the document

The rest of this document contains more detailed information about the components and development process of RRComSSys, and is laid out in six more chapters and an appendix chapter.

Chapter 2: Chapter 2 covers the organization of the project in terms of the distribution of the tasks among the team members, as well as expressing milestones throughout the project, and what expectations for delivery are associated with each milestone. Furthermore, the hardware and software requirements to develop the system, as well as the team roles are breakdown.

Chapter 3: Chapter 3 covers the requirements and elicitation analysis. High level functionality will be described here for each of the 8 use cases.

Chapter 4: This chapter presents the overall system design. The overview consists of a package diagram which illustrates the major subsystems and gives an overview of each subsystem. Secondly the Meta-model for the DSL is explained in detail. As in the previous document the UML profiles and generative architecture are described. Last but not least in this chapter is the validation of the system model. The test cases will be displayed in a tabular form here.

Chapter 5: Object design is fundamental to our system. Diagrams in this chapter include the minimal class diagram, sequence diagrams, state machines and more. This chapter is finished off with the validation of the system model. A checklist will be displayed for the artifacts. Lastly a structured walkthrough will be performed on the artifacts and recorded in tabular form.

Chapter 6: The Implementation chapter will describe the platform specific model used to run the system on. The validation of the system’s results will be recorded and displayed in tabular form.

Chapter 7: This chapter includes a glossary of terms used in this document for reader not specifically versed in the terminology of the implementation.

Chapter 8: This last chapter contains appendices consisting of supporting documentation and visual aids for the previous chapters.

# Project Plan

This section presents the general organizational structure of the RRComSSys development plan. This includes the various work distributions during each phase of the project, the hardware and software requirements to develop and test the system, as well as milestones and deliverables produced during each stage.

## Project organization – assignment of roles for the entire project

The following figures display the organizational structure of the roles involved in the RRComSSys project for each of the three phases of development:

Software Requirements and Analysis Document:

Project Leader

Jean Rodriguez

VModel Developer

Marcelo Lopez

Transform Developer

Jean Rodriguez

Synthesis/Workflow Engine Developer

Denis Antoine

Synthesis/Workflow Engine Developer – Harika Chirumamilla

Minute Taker – Harika

Chirumamilla

Design Document:

Project Leader

Denis Antoine

VModel Developer

Marcelo Lopez

Transform Developer

Jean Rodriguez

Synthesis/Workflow Engine Developer

Denis Antoine

Synthesis/Workflow Engine Developer – Harika Chirumamilla

Minute Taker – Harika

Chirumamilla

Final Deliverable:

Project Leader

Marcelo Lopez

VModel Developer

Marcelo Lopez

Transform Developer

Jean Rodriguez

Synthesis/Workflow Engine Developer

Denis Antoine

Synthesis/Workflow Engine Developer – Harika Chirumamilla

Minute Taker – Harika

Chirumamilla

Leader: Organize and appropriately distribute the tasks according to members’ area of expertise; ensures that all milestones are met accordingly and with sufficient time to complete corrections as complete for the professor.

VModel Developer: Creates the Visual Modeling Environment within the Eclipse framework. This refers to the creation of the Meta-model, or GCML, by developing all the necessary visual entities to produce the communication models, bound by the designer’s constraints.

Transformer Developer: The Transformer developer will create the necessary classes and infrastructure necessary to ensure that the transformations, and subsequent validation and parsing of the G-CML and XCML models is proper and meets the requirements of the modelers designs.

SE/Workflow Developer: The Synthesis/Workflow Engine developers create the classes that will implement the core entities of both the communication models, and the workflow infrastructure. Also, they develop those classes that bridge to the communication services (Skype and Google Talk) APIs.

Minute/Note Taker: Keeps record of the details of every meeting, including time and date, attendance, and the topics discussed during each meeting. This information is distributed to the rest of the group by the end of each week.

## Hardware and Software Requirements

**Hardware needed**

Processor: Intel or AMD 2Ghz or faster

Memory: 1 GB of RAM (minimum), 2Gb (recommended)

Hard Drive: 80 GB

Headset with built-in microphone (optional)

Webcam (optional)

Internet Connection: Cable/DSL or higher

**Software needed**

Windows XP Professional / Windows Vista

Microsoft Word 2007

Microsoft Visio / Star UML

Microsoft Project 2007

Eclipse 3.4.1 IDE

Skype Java API

Google Talk/Smack API (optional, but recommended)

       Skype4Com Library

       Microsoft .NET Framework 3.5

       Microsoft Visual Studio 2008 Express

       Skype application software

**2.3 Work Breakdown**

Project Schedule: Refer to Project Plan in Appendix H

Milestone 1 consists of the completion of the Use Case specification and Analysis Phase. This results in a software requirements document handed in to the professor. This deliverable also includes and should cover the necessary object models.

Milestone 2 consists of the completion of Design Phase, the creation of the Meta model and the design and implementation of the RRComSSys in general. This includes the implementation of the parsers and the validators as well. The result of this phase is presented in a design document handed in to professor.

Finally, Milestone 3, which is to say the final deliverable of RRComSSys will consist of the final tasks to complete the system and the testing/validation phase. The final deliverable document includes the two previous deliverable plus the details of this last milestone.

# Requirements elicitation and analysis

## Overview

This chapter defines the requirements obtained by the use cases as well as provides use case diagrams for the system. The use cases will provide the functionality of the system through general event sequences describing a variety of interactions between actor and the system. The functional requirements described the functionality at a higher level for the 8 use cases to be implemented. The next section describes the user level requirements not directly related to functionality. The system model shows both a static and dynamic view of the system. Also the use case model, object model dynamic model will be laid out. The last section before the validation contains the user interfaces. Validation of the analysis model will contain the 24 test cases and results from the testing will be lain out in tabular form.

## Functional Requirements

The system shall:

1. Allow health care professionals to create communication models based on the CML language.

Use cases: All use cases related to General Connection.

Use Case ID(s): C01\_GeneralConnection, C02\_TextOnlyConnection, C03\_TextAudioConnection and C04\_Audio Video Connection.

1. Allow health care professionals to communicate with one or many users using audio, video and text conversations.

Use cases: All use cases related to General Connection.

Use Case ID(s): C01\_GeneralConnection, C02\_TextOnlyConnection, C03\_TextAudioConnection and C04\_Audio Video Connection.

1. Allow health care professionals to establish a conversation with either Skype or GoogleTalk.

Use cases: All use cases related to General Connection.

Use Case ID(s): C01\_GeneralConnection, C02\_TextOnlyConnection, C03\_TextAudioConnection and C04\_Audio Video Connection.

1. Allow health care professionals to use a workflow model to establish communication models that can execute alternate conversation activities.

Use cases: All use cases related to General Connection and Workflow Connection.

Use Case ID(s): C01\_GeneralConnection, C02\_TextOnlyConnection, C03\_TextAudioConnection, C04\_Audio Video Connection, W01\_WorkflowWithAlternate and W02\_WorkflowWithRetry.

1. Allow health care professionals to use workflow to build communication models that can try a conversation activity repeatedly until a connection is established.

Use cases: All use cases related to General Connection and Workflow Connection.

Use Case ID(s): C01\_GeneralConnection, C02\_TextOnlyConnection, C03\_TextAudioConnection, W01\_WorkflowWithAlternate and W02\_WorkflowWithRetry.

1. Allow health care professionals to send and receive files within a conversation. Use cases: All use cases related to File Transfer.

## Non functional requirements

**Non functional Requirements:**

* Usability: The system requires no training (see all use cases in appendix B).
* Reliability:
* The system shall at most have 5% failures for every twenty four hours of operation (see all use cases in appendix B).
* The system shall perform function for 100% of all sessions (RRComSSys\_TextOnlyConnection, RRComSSys\_TextAudioConnection, RRComSSys \_AudioVideoConnection, RRComSSys\_SendFile, RRComSSys\_ReceiveFile, F08\_SecurityFileTransfer, W01\_WorkflowWithAlternateConnection, RRComSSys\_Workflow with Retry in appendix B).
* Performance:
  + The system shall perform function with 5 seconds (see use cases RRComSSys\_GeneralConnection,RRComSSys\_TextOnlyConnection,RRComSSys\_TextAudioConnection, RRComSSys\_AudioVideoConnection, RRComSSys\_Logout, S01\_BlockList, S02\_PrivateConnection, W01\_WorkFlowWithAlternateConnection, RRComSSys\_Workflow )
  + The system shall perform function in less than 1 second (see use case S04\_AutomaticLogout).
  + The system shall perform function in 5 minutes (see use case L01)
* Supportability: The system shall be able to load in Internet Explorer 5 or higher and Mozilla Firefox version 2.5 or higher (see all use cases in appendix B).

## System Models

The system models presented in this section represent the static and the dynamic views of the system: the class diagram represent the system’s static structure while the use cases, sequence diagrams, and samples of the user interface represent the system’s behavior.

### Use case model (See Appendix A)

This section mainly refers to Appendix A. Figure 8.1.1 is a Use Case Diagram From the perspective of the end user. Illustrated here is basically the CVM Environment and the Communication Model attached to the user. The next figure namely, figure 8.1.2 is the CVM Environment Use Case Diagram illustrates all use cases from the users point of view. Use cases which are extended are represented by branching off from their “super” use case. Finally figure 8.1.3 represents the Communication Modeling Use Case Diagram. It illustrates the connection types. There are a total of 5 connection types. All connection types are a subset of the general connection. The *WorkflowWithRetry* is the only connection which extends a connection (namely the *WorkflowWithAlt* connection).

### Object model (See Appendix C)

This section discusses the details of the Object model for RRComSSys. That includes a description of the subsystem interfaces, the component classes, and other objects that will be designed to then implement the system from. A detailed class diagram and object diagrams for the use cases that interact with the API are presented following.

The RRComSSys contains four subsystems: Visual Environment (that is to say the component which allows the user to create a model of the communication session or sessions he or she wishes to establish), a Model Transformer which converts the native GMF format representation for the visual model into a grammatically equivalent model (in XML) for the communication session. A Schema Transformation Environment, and Synthesis Engine, which transform this communication grammar into an equivalent grammar which can be validated and executed by the CVM. Following, each of these components is revealed along with the classes that comprise each subsystem.

The class diagram depicted have been designed on the basis of the requirements described by the stakeholder for the project, and following the general architecture of the system by the development team. The schema used follows the schema guidelines provided by the project facilitator. Here is a description of the diagram:

Component 1: Visual Environment (VE) in GMF – This portion will be implemented as a specialized OSGi modeling workbench application based on the Eclipse GMF, and in and of itself does not comprise actual code ( though it’s specification is implied by the underlying visual model set forth by the developers).

Component 2: Model Transformer (MT):

The *ModelTransformer* class which will be implemented essentially takes on the responsibility of transforming the G-CML model described in the VE into an equivalent X-CML model.

The *FileSystemHelper* class – A helper class used by the *ModelTransformer* class to retrieve the files generated by the VE (Eclipse), and to save the XML files generated by the software.

The *ModelValidator* class – is used by the *ModelTransformer* class and the *SchemaTransformer* classes to perform validation against the underlying schemas of the models generated by the.

*ModelParser* class - used by the *ModelTransformer* class and the *SchemaTransformer* class to parse the models generated by our software.

Component 3: Schema Transformation Environment (STE)

*SchemaTransformer* class- used to implement the validation of the communication model for realization. This class will use the helper classes described above.

Component 4: Synthesis Engine (SE)

*SynthesisEngine* class - used to implement the realization of the X-CML communication model by making the appropriate calls to the Skype and Smack APIs. This class will use the following classes: *Person*, *isAttached*, *Device*, *Connection*, *capabilityType*, *mediumType*.

### Dynamic model

The sequence diagrams depicted in appendix D represent the system's behavior in terms of the interactions and message flow between and among the different objects within the system. Following, is a brief description of each of the diagrams included. Figure 8.4.1 Text Connection shows the sequence of events that occur when the user requests to instantiate a simple text based conversation with another party. The user selects the model to execute, and provides the participant id for the party to connect to, whereupon the CVM interface will pass the supplied information to the Controller portion of the CVM which will be responsible for the establishing the connection via the Communications Object components. Once a connection is established the controller activates the type of communication capability within the Communication Object Components. At some point, the user elects to terminate the connection, and the controller passes a disconnect message to the Communications Object Component to disconnect the session, and the session is terminated.

Figure 8.4.2 Text/Audio Connection shows the sequence of events that occur when the user requests to instantiate a simple text based conversation with another party. The user selects a model which support both text and audio communication protocols, to execute, and provides the participant id for the party to connect to, whereupon the CVM interface will pass the supplied information to the Controller portion of the CVM which will be responsible for the establishing the connection via the Communications Object components. Once a connection is established the controller requests activation of both text and audio communication protocol capabilities within the Communication Object Components. At some point, the user elects to terminate the connection, and the controller passes a disconnect message to the Communications Object Component to disconnect the session, and the session is terminated.

Figure 8.4.3 Audio/Video Connection shows the sequence of events that occur when the user requests to instantiate a simple text based conversation with another party. The user selects a model which support both text and audio communication protocols, to execute, and provides the participant id for the party to connect to, whereupon the CVM interface will pass the supplied information to the Controller portion of the CVM which will be responsible for the establishing the connection via the Communications Object components. Once a connection is established the controller requests activation of both audio and video communication protocol capabilities within the Communication Object Components. At some point, the user elects to terminate the connection, and the controller passes a disconnect message to the Communications Object Component to disconnect the session, and the session is terminated.

Figure 8.4.4 Receive File illustrate the sequence of events that occur when the user has an established connection with another user and has been presented with a request accept a file from the other party. The elects to accept the file transfer, where upon the CVM will relay the message to the Controller which notifies the Communication Object Components to inform the other parties’ CVM to initiate the file transfer. Once the file transfer process has been negotiated, the user is informed visually that a file transfer is underway. Once the file has been completed, the user is again notified that file has been receive, and then the user selects the file to view the received file, whereupon the CVM will display the location of the received file.

Figure 8.4.5 Send File depicts the scenario where the user wishes to send a file to another party, and instantiates the model to do so. And after providing the proper participant id, and a connection is established, selects the file to transfer to the other user, and the CVM initiates a file transfer process. Once the other user elects to accept (see the previous scenario description), the CVM will perform the file transfer itself. Once the file has been transferred, the user then elects to terminate the call, and the CVM disconnects from the communication service.

Figure 8.4.6 Send Secure File depicts the scenario where the user wishes to send a file to another party, and instantiates the model to do so. The difference between this scenario ad the previous one is that a secure connection tunnel is requested to ensure that the data exchange occurs in via encryption of the data stream.

Figure 8.4.7 Workflow Alternate shows the sequence of events where the CVM’s workflow engine handles a scenario where a user attempts to establish a conversation, and in the event that the first conversation is unable to be connected, will failover and attempt to connect via an alternate conversation.

Figure 8.4.8 Workflow with Retry depicts the same failover functionality as “Workflow Alternate” except the graphical model was designed in a manner where the failover conversation is the original conversation itself, so the CVM will retry to establish a connection. Once a connection is established, the text protocol is activated, and the user is able to send messages to the other party until they choose to terminate the conversation.

### User Interfaces

Modeling Environment – The Modeling Environment is used to create models that get saved as a gcml file. The gcml file is later used to realize the communication. See Figure E1, E2, E3 in Appendix E.

Login Screen – The Login Screen is the entry gate to the CVM system. In order to load a model you first have to provide a valid username and password. See Figure E4 in Appendix E.

File Loader – The File Loader Form allows loading a gcml file and providing an output xcml file. Clicking load will execute the model provided. See Figure E5 in Appendix E.

Choose File Dialog – The Choose File Dialog allows you to choose an input gcml file and an output xcml file. See Figure E6 in Appendix E.

Missing Attribute Form – The Missing Attribute Form allows you to enter any missing attributes from the gcml model provided. See Figure E7 in Appendix E.

Skype – The Skype interface is where any calls, file transfers and other communication instances are realized. See Figure E8 in Appendix E.

## 3.5 Validation of the analysis model

This section describes the procedures followed while evaluating the system to determine if the artifacts in the analysis model satisfy the defined requirements.



### Test Cases

The following test cases have been developed to examine the accuracy of the use cases.

**Test Case 1**: Purpose – Set up a text connection

|  |  |
| --- | --- |
| **Test Case ID** | TS01- Create\_Text\_Conn |
| **Path Type** | Sunny Day Scenario #1 |
| **Preconditions** | The GCML Diagram Editor is Open  The user has selected File -> New -> GCML Diagram  The user give the diagram a new title followed by the file extension of .gcml\_diagram  An empty canvas is visible |
| **Input** | Invoke: The user drags an Participant node onto the canvas  The user then keys in their ID, Name, And Role  Invoke: The user drags an Device node onto the canvas  The user then adds the text capability to the device  The user gives the device a name of “text”  Invoke: The user drags an isAttached node onto the canvas  The user then drags an arrow from isAttached to the Participant node.  Invoke: The user drags another Participant node onto the canvas  The user then keys in the ID, Name, And Role of the second user  Invoke: The user drags an Device node onto the canvas representing the second user  The user then adds the text capability to this device  The user gives the device a name of “text2”  Invoke: The user drags an isAttached node onto the canvas  The user then drags an arrow from isAttached to the newly created Participant node.  Invoke: The user drags an Connection node onto the canvas  The user then keys in the bandwidth and connection id.  The user then connects both devices to the connection.  The user clicks the Save icon to save the diagram. The user then launches the CVM application. The user is prompted for a password and logon name. The user enters their user name “denis” and password of “password”. The user is successfully logged on. The user Loads the GCML file created. The GCML loads successfully without any errors and than the users selects the execute button to execute the model. |
| **Expected Output** | A file by the name of YourFileNameHere.gcml\_diagram and also a Text Connection |

|  |  |
| --- | --- |
| **Test Case ID** | TS01- Create\_Text\_Conn |
| **Path Type** | Alternative Scenario #1 |
| **Preconditions** | The GCML Diagram Editor is Open  The user has selected File -> New -> GCML Diagram  An empty canvas is visible |
| **Input** | Invoke: The user drags an Participant node onto the canvas  The user then keys in their ID, Name, And Role  Invoke: The user drags an Device node onto the canvas  The user then adds the text capability to the device  The user gives the device a name of “text”  Invoke: The user drags an isAttached node onto the canvas  The user then drags an arrow from isAttached to the Participant node.  Invoke: The user drags another Participant node onto the canvas  The user DOES NOT key in the ID, Name, And Role of the second user. Invoke: The user drags an Device node onto the canvas representing the second user  The user then adds the text capability to this device  The user gives the device a name of “text2”  Invoke: The user drags an isAttached node onto the canvas  The user then drags an arrow from isAttached to the newly created Participant node.  Invoke: The user drags an Connection node onto the canvas  The user then keys in the bandwidth and connection id.  The user then connects both devices to the connection.  The user clicks the Save icon to save the diagram. The user then launches the CVM application. The user is prompted for a password and logon name. The user enters their user name “harika” and password of “password”. The user is successfully logged on. The user Loads the GCML file created. The GCML does not validate. |
| **Expected Output** | A file by the name of YourFileNameHere.gcml\_diagram and the “Missing Attribute” Form requesting the appropriate participant fields namely: ID, Name, and Role. The error log displays the various error messages. |

|  |  |
| --- | --- |
| **Test Case ID** | TS01- Create\_Text\_Conn |
| **Path Type** | Alternative Scenario #2 |
| **Preconditions** | The GCML Diagram Editor is Open  The user has selected File -> New -> GCML Diagram  An empty canvas is visible |
| **Input** | Invoke: The user drags an Participant node onto the canvas  The user then keys in their ID, Name, And Role  Invoke: The user drags an Device node onto the canvas  The user then adds the text capability to the device  The user gives the device a name of “text”  Invoke: The user drags an isAttached node onto the canvas  The user then drags an arrow from isAttached to the Participant node.  Invoke: The user drags another Participant node onto the canvas  The user keys in the ID, Name, And Role of the second user. Invoke: The user drags a Device node onto the canvas for the second user. The user does not add any capability to this device.  The user gives the device a name of “text2”  Invoke: The user drags an isAttached node onto the canvas  The user then drags an arrow from isAttached to the newly created Participant node.  Invoke: The user does not include a connection node.  The user clicks the Save icon to save the diagram. The user then launches the CVM application. The user is prompted for a password and logon name. The user enters their user name “jean” and password of “password”. The user is successfully logged on. The user Loads the GCML file created. The GCML does not validate and the error log displays a list of errors. |
| **Expected Output** | A file by the name of YourFileNameHere.gcml\_diagram and the error log displays a list of errors. |

**Test Case 2**: Purpose – Set up an audio connection

|  |  |
| --- | --- |
| **Test Case ID** | TS02- Create\_Audio\_Conn |
| **Path Type** | Sunny Day Scenario #1 |
| **Preconditions** | The GCML Diagram Editor is Open  The user has selected File -> New -> GCML Diagram  The user give the diagram a new title followed by the file extension of .gcml\_diagram  An empty canvas is visible |
| **Input** | Invoke: The user drags an Participant node onto the canvas  The user then keys in their ID, Name, And Role  Invoke: The user drags an Device node onto the canvas  The user then adds the Audio capability to the device  The user gives the device a name of “audio”  Invoke: The user drags an isAttached node onto the canvas  The user then drags an arrow from isAttached to the Participant node. The user then drags an arrow from isAttached to the Device node.  Invoke: The user drags another Participant node onto the canvas  The user then keys in the ID, Name, And Role of the second user  Invoke: The user drags an Device node onto the canvas representing the second user  The user then adds the Audio capability to this device  The user gives the device a name of “audio2”  Invoke: The user drags an isAttached node onto the canvas  The user then drags an arrow from isAttached to the Participant node. The user then drags an arrow from isAttached to the Device node.  Invoke: The user drags an Connection node onto the canvas  The user then keys in the bandwidth and connection id.  The user then connects both devices to the connection.  The user clicks the Save icon to save the diagram. The user then launches the CVM application. The user is prompted for a password and logon name. The user enters their user name “denis” and password of “password”. The user is successfully logged on. The user Loads the GCML file created. The GCML loads successfully without any errors and than the users selects the execute button to execute the model. |
| **Expected Output** | A file by the name of YourFileNameHere.gcml\_diagram and a Audio Connection. |

|  |  |
| --- | --- |
| **Test Case ID** | TS02- Create\_Audio\_Conn |
| **Path Type** | Alternate Scenario #1 |
| **Preconditions** | The GCML Diagram Editor is Open  The user has selected File -> New -> GCML Diagram  The user give the diagram a new title followed by the file extension of .gcml\_diagram  An empty canvas is visible |
| **Input** | Invoke: The user drags an Participant node onto the canvas  The user then keys in their ID, Name, And Role  Invoke: The user drags an Device node onto the canvas  The user then adds the Audio capability to the device  The user gives the device a name of “audio”  Invoke: The user drags an isAttached node onto the canvas  The user then drags an arrow from isAttached to the Participant node. The user then drags an arrow from isAttached to the Device node.  Invoke: The user drags another Participant node onto the canvas  The user DOES NOT key in the ID, Name, And Role of the second user. Invoke: The user drags an Device node onto the canvas representing the second user  The user then adds the Audio capability to this device  The user gives the device a name of “audio2”  Invoke: The user drags an isAttached node onto the canvas  The user then drags an arrow from isAttached to the Participant node. The user then drags an arrow from isAttached to the Device node.  Invoke: The user drags an Connection node onto the canvas  The user then keys in the bandwidth and connection id.  The user then connects both devices to the connection.  The user clicks the Save icon to save the diagram. The user then launches the CVM application. The user is prompted for a password and logon name. The user enters their user name “denis” and password of “password”. The user is successfully logged on. The user Loads the GCML file created. The GCML does not validate. |
| **Expected Output** | A file by the name of YourFileNameHere.gcml\_diagram and an “Missing Attribute” Form requesting the appropriate participant fields namely: ID, Name, and Role. The error log displays the various error messages. |

|  |  |
| --- | --- |
| **Test Case ID** | TS02- Create\_Audio\_Conn |
| **Path Type** | Alternate Scenario #2 |
| **Preconditions** | The GCML Diagram Editor is Open  The user has selected File -> New -> GCML Diagram  The user give the diagram a new title followed by the file extension of .gcml\_diagram  An empty canvas is visible |
| **Input** | Invoke: The user drags an Participant node onto the canvas  The user then keys in their ID, Name, And Role  Invoke: The user drags an Device node onto the canvas  The user then adds the Audio capability to the device  The user gives the device a name of “audio”  Invoke: The user drags an isAttached node onto the canvas  The user then drags an arrow from isAttached to the Participant node. The user then drags an arrow from isAttached to the Device node.  Invoke: The user drags another Participant node onto the canvas  The user DOES NOT key in the ID, Name, And Role of the second user. Invoke: The user drags an Device node onto the canvas representing the second user  The user then adds the Audio capability to this device  The user gives the device a name of “audio2”  Invoke: The user drags an isAttached node onto the canvas  The user then drags an arrow from isAttached to the Participant node. The user then drags an arrow from isAttached to the Device node.  Invoke: The user does not include a connection node.  The user clicks the Save icon to save the diagram. The user then launches the CVM application. The user is prompted for a password and logon name. The user enters their user name “denis” and password of “password”. The user is successfully logged on. The user Loads the GCML file created. The GCML does not validate. |
| **Expected Output** | A file by the name of YourFileNameHere.gcml\_diagram and the error log displays a list of errors. |

**Test Case 3**: Purpose – Set up an audio and video connection

|  |  |
| --- | --- |
| **Test Case ID** | TS03- Create\_AudioVideo\_Conn |
| **Path Type** | Sunny Day Scenario #1 |
| **Preconditions** | The GCML Diagram Editor is Open  The user has selected File -> New -> GCML Diagram  The user give the diagram a new title followed by the file extension of .gcml\_diagram  An empty canvas is visible |
| **Input** | Invoke: The user drags an Participant node onto the canvas  The user then keys in their ID, Name, And Role  Invoke: The user drags an Device node onto the canvas  The user then adds the Audio and Video capability to the device  The user gives the device a name of “AudioVideo”  Invoke: The user drags an isAttached node onto the canvas  The user then drags an arrow from isAttached to the Participant node. The user then drags an arrow from isAttached to the Device node.  Invoke: The user drags another Participant node onto the canvas  The user then keys in the ID, Name, And Role of the second user  Invoke: The user drags an Device node onto the canvas representing the second user  The user then adds the Audio and Video capability to this device  The user gives the device a name of “AudioVideo2”  Invoke: The user drags an isAttached node onto the canvas  The user then drags an arrow from isAttached to the Participant node. The user then drags an arrow from isAttached to the Device node.  Invoke: The user drags an Connection node onto the canvas  The user then keys in the bandwidth and connection id.  The user then connects both devices to the connection.  The user clicks the Save icon to save the diagram. The user then launches the CVM application. The user is prompted for a password and logon name. The user enters their user name “denis” and password of “password”. The user is successfully logged on. The user Loads the GCML file created. The GCML loads successfully without any errors and than the users selects the execute button to execute the model. |
| **Expected Output** | A file by the name of YourFileNameHere.gcml\_diagram and also an Audio Video connection is established. |

|  |  |
| --- | --- |
| **Test Case ID** | TS03- Create\_AudioVideo\_Conn |
| **Path Type** | Alternate Scenario #1 |
| **Preconditions** | The GCML Diagram Editor is Open  The user has selected File -> New -> GCML Diagram  The user give the diagram a new title followed by the file extension of .gcml\_diagram  An empty canvas is visible |
| **Input** | Invoke: The user drags an Participant node onto the canvas  The user then keys in their ID, Name, And Role  Invoke: The user drags an Device node onto the canvas  The user then adds the Audio and Video capability to the device  The user gives the device a name of “AudioVideo”  Invoke: The user drags an isAttached node onto the canvas  The user then drags an arrow from isAttached to the Participant node. The user then drags an arrow from isAttached to the Device node.  Invoke: The user drags another Participant node onto the canvas  The user then keys in the ID, Name, And Role of the second user  Invoke: The user drags an Device node onto the canvas representing the second user  The user then adds the Audio and Video capability to this device  The user gives the device a name of “AudioVideo 2”  Invoke: The user drags an isAttached node onto the canvas  The user then drags an arrow from isAttached to the Participant node. The user then drags an arrow from isAttached to the Device node.  Invoke: The user drags an Connection node onto the canvas  The user then keys in a connection id.  The user then connects both devices to the connection.  The user clicks the Save icon to save the diagram. The user then launches the CVM application. The user is prompted for a password and logon name. The user enters their user name “denis” and password of “password”. The user is successfully logged on. The user Loads the GCML file created. The GCML does not validate due to the missing attributes. |
| **Expected Output** | A file by the name of YourFileNameHere.gcml\_diagram and also Missing Attribute” Form requesting the appropriate connection field namely: the bandwidth. Errors are displayed on the error log. |

|  |  |
| --- | --- |
| **Test Case ID** | TS03- Create\_AudioVideo\_Conn |
| **Path Type** | Alternate Scenario #2 |
| **Preconditions** | The GCML Diagram Editor is Open  The user has selected File -> New -> GCML Diagram  The user give the diagram a new title followed by the file extension of .gcml\_diagram  An empty canvas is visible |
| **Input** | Invoke: The user drags an Participant node onto the canvas  The user then keys in their ID, Name, And Role  Invoke: The user drags an Device node onto the canvas  The user then adds the Audio and Video capability to the device  The user gives the device a name of “AudioVideo”  Invoke: The user drags an isAttached node onto the canvas  The user then drags an arrow from isAttached to the Participant node. The user then drags an arrow from isAttached to the Device node.  Invoke: The user drags another Participant node onto the canvas  The user then keys in the ID, Name, And Role of the second user  Invoke: The user drags an Device node onto the canvas representing the second user  The user then adds the Audio and Video capability to this device  The user gives the device a name of “AudioVideo 2”  Invoke: The user drags an isAttached node onto the canvas  The user then drags an arrow from isAttached to the Participant node. The user then drags an arrow from isAttached to the Device node.  Invoke: The user then launches the CVM application. The user is prompted for a password and logon name. The user enters their user name “denis” and password of “password”. The user is successfully logged on. The user Loads the GCML file created. The GCML does not validate due to the missing connection node. |
| **Expected Output** | A file by the name of YourFileNameHere.gcml\_diagram errors are displayed on the error log. |

**Test Case 4**: Purpose – Receive a File from another user

|  |  |
| --- | --- |
| **Test Case ID** | TS04- F03\_ReceiveFile |
| **Path Type** | Sunny Day Scenario #1 |
| **Preconditions** | The Sender and Receiver are connected  The Sender has sent the file  Receiver has accepted the file sent |
| **Input** | Invoke: Accept file dialog |
| **Expected Output** | Message: “File transferred successfully”. |

|  |  |
| --- | --- |
| **Test Case ID** | TS04- F03\_ReceiveFile |
| **Path Type** | Alternative Scenario #2 |
| **Preconditions** | The Sender and Receiver are connected  The Sender has sent the file  Receiver has accepted the file sent |
| **Input** | Invoke: Decline File Button |
| **Expected Output** | Message: “File transferred declined”. |

|  |  |
| --- | --- |
| **Test Case ID** | TS04- F03\_ReceiveFile |
| **Path Type** | Alternative Scenario #3 |
| **Preconditions** | The Sender and Receiver are connected  The Sender has sent the file  Receiver has accepted the file sent |
| **Input** | Invoke: Accept file dialog  Receiver or Sender disconnects |
| **Expected Output** | Message: “File transferred failed”. |

**Test Case 5**: Purpose – Send a file from one user to another user

|  |  |
| --- | --- |
| **Test Case ID** | TS05- F01\_SendFile |
| **Path Type** | Sunny Day Scenario #1 |
| **Preconditions** | The Sender and Receiver are connected  The sender has the Send file capability |
| **Input** | Invoke: Select the send file option  Click Ok to confirm the process |
| **Expected Output** | Message: “File transferred successfully”. |

|  |  |
| --- | --- |
| **Test Case ID** | TS05- F01\_SendFile |
| **Path Type** | Alternative Scenario #2 |
| **Preconditions** | The Sender and Receiver are connected  The sender has the Send file capability |
| **Input** | Invoke: Select the send file option  Click Ok to confirm the process  Sender selects the cancel Button to cancel the transfer |
| **Expected Output** | Message: “File transferred canceled”. |

|  |  |
| --- | --- |
| **Test Case ID** | TS05- F01\_SendFile |
| **Path Type** | Alternative Scenario #3 |
| **Preconditions** | The Sender and Receiver are connected  The sender has the Send file capability |
| **Input** | Invoke: Select the send file option  Click Ok to confirm the process  File is too large |
| **Expected Output** | Message: “File is too large to be sent”. |

**Test Case 6**: Purpose – Securely transfer a file from one user to another user

|  |  |
| --- | --- |
| **Test Case ID** | TS06- F08\_SecurityFileTransfer |
| **Path Type** | Sunny Day Scenario #1 |
| **Preconditions** | The Sender and Receiver are connected  The receiver and sender have agreed to a password |
| **Input** | Invoke: Select the send secure file option  Sender keys in the password  Sender gets a dialog requesting the password before initiating the transfer  Receiver keys in the password  Click Ok to confirm the process  File Transfer Begins |
| **Expected Output** | Message: “File transferred successfully”. |

|  |  |
| --- | --- |
| **Test Case ID** | TS06- F08\_SecurityFileTransfer |
| **Path Type** | Alternative Scenario #2 |
| **Preconditions** | The Sender and Receiver are connected  The receiver and sender have agreed to a password |
| **Input** | Invoke: Select the send secure file option  Sender keys in the password  Receiver gets a dialog requesting the password before initiating the transfer  Receiver keys in the password  Click Ok to confirm the process  Passwords Don’t Match |
| **Expected Output** | Message: “Incorrect Password”. |

|  |  |
| --- | --- |
| **Test Case ID** | TS06- F08\_SecurityFileTransfer |
| **Path Type** | Alternative Scenario #3 |
| **Preconditions** | The Sender and Receiver are connected  The receiver and sender have agreed to a password |
| **Input** | Invoke: Select the send secure file option  Sender keys in the password  Password is shorter than 5 characters |
| **Expected Output** | Message: “Password must be greater than 4 characters”. |

**Test Case 7**: Purpose – to build communication models that can try a conversation

Activity repeatedly until a connection is established

|  |  |
| --- | --- |
| **Test Case ID** | TS07-W02\_Workflow with Retry |
| **Path Type** | Sunny Day Scenario #1 |
| **Preconditions** | The user must be logged on, a new file must be open and the canvas is set and ready to accept the new communication model. |
| **Input** | Invoke: The user sets up a workflow conversation as described in use case W01\_WorkflowWithAlternateConnection.  Next, the user connects the “DecisionPoint” shape to the “ConversationActivity” shape in which another user is the other participant, as described in use case W01\_WorkflowWithAlternateConnection. |
| **Expected Output** | The canvas now holds a communication model that can be executed. When executed a connection is attempted between both users. If that connection is successful, the connection is established. If that connection fails, the connection between the first user and a third user is attempted. If the connection between the users also fails, the connections are retried until one of the connections is successful.  Message: “Connection Successful”. |

|  |  |
| --- | --- |
| **Test Case ID** | TS07-W02\_Workflow with Retry |
| **Path Type** | Alternative Scenario #2 |
| **Preconditions** | Invoke: The user sets up a workflow conversation as described in use case W01\_WorkflowWithAlternateConnection.  Next, the user connects the “DecisionPoint” shape to the “ConversationActivity” shape in which another user is the other participant, as described in use case W01\_WorkflowWithAlternateConnection. A condition is met(too much time has passed or an exception has occurred or the connection(s) was/were attempted X amount of times). |
| **Input** | Invoke: The user sets up a workflow conversation as described in use case W01\_WorkflowWithAlternateConnection.  Next, the user connects the “DecisionPoint” shape to the “ConversationActivity” shape in which another user is the other participant, as described in use case W01\_WorkflowWithAlternateConnection. |
| **Expected Output** | Message: “Both Connections can not be established”. |

|  |  |
| --- | --- |
| **Test Case ID** | TS07-W02\_Workflow with Retry |
| **Path Type** | Alternative Scenario #3 |
| **Preconditions** | The user must be logged on, a new file must be open and the canvas is set and ready to accept the new communication model. |
| **Input** | Invoke: The user sets up a workflow conversation as described in use case W01\_WorkflowWithAlternateConnection.  Next, the user connects the “DecisionPoint” shape to the “ConversationActivity” shape in which another user is the other participant, as described in use case W01\_WorkflowWithAlternateConnection. |
| **Expected Output** | The canvas now holds a communication model that can be executed. When executed a connection is attempted between both users. The 1st connection is unsuccessful, but the 2nd connection is established. The first connection is not established.  Message: “Connection with user 1 was unsuccessful; connection with user 2 was successful”. |

**Test Case 8**: Purpose – Allow health care professionals to use a workflow model to establish communication models that can execute alternate conversation activities.

|  |  |
| --- | --- |
| **Test Case ID** | TS08-W01\_WorkflowWithAlternateConnection |
| **Path Type** | Sunny Day Scenario #1 |
| **Preconditions** | The user logged into the system and creates a new canvas. A new canvas is set and ready to accept the new communication model. |
| **Input** | The user drags a flow bar to establish a connection to first user after the connection is established then the flow bar ends. |
| **Expected Output** | Message: “Connection successful”. |

|  |  |
| --- | --- |
| **Test Case ID** | TS08-W01\_WorkflowWithAlternateConnection |
| **Path Type** | Alternative Scenario #2 |
| **Preconditions** | The user logged into the system and creates a new canvas. A new canvas is set and ready to accept the new communication model. |
| **Input** | The user drags a flow bar and tries to establish a connection to the first user and the connection is not established then the flow bar continues then user tries to establish the connection the other user then the flow bar ends after success. |
| **Expected Output** | Message: “Connection Failed; Connection Successful”. |

|  |  |
| --- | --- |
| **Test Case ID** | TS08-W01\_WorkflowWithAlternateConnection |
| **Path Type** | Alternative Scenario #3 |
| **Preconditions** | The user logged into the system and creates a new canvas. A new canvas is set and ready to accept the new communication model. |
| **Input** | The user drags a flow bar and tries to establish a connection to first user and the connection is not established then the flow bar continues and then user tries to establish the connection to another user which is also unsuccessful. |
| **Expected Output** | Message: “Connection Failed; Connection Failed”. |

### Checklist

The following checklist is used to inspect the completeness, correctness, and consistency of the design model.

Use Case Diagram

|  |  |  |
| --- | --- | --- |
|  | YES | NO |
| 1- Are all actors defined in the model? | X |  |
| 2- Has all the functionality been described in the model? | X |  |
| 3- Is the output from the system defined for every input from the actor, both for normal flow of events and variations? | X |  |
| 4- Have all the variants to the normal flow of events been identified? | X |  |
| 5- Is the functionality associated with each participating actor correct? | X |  |
| 6- Are the *include* and *extend* associations defined correctly? | X |  |
| 7- Are all the naming schemes the same? | X |  |
| 8- Are all the font styles used the same? | X |  |

Sequence Diagram

|  |  |  |
| --- | --- | --- |
|  | YES | NO |
| 1- Does the diagram include all the objects that participate during the execution of this functionality? | X |  |
| 2- Does the diagram include any object that does not participate during the execution of this functionality? |  | X |
| 3- Does the participating actor have access to invoke this action? | X |  |
| 4- Does each object provide the methods shown in the diagram as being invoked? | X |  |
| 5- Does the system follow the same flow of control every time the same type of input is provided? | X |  |
| 6- Does the diagram include all possible alternative flows of control? | X |  |
| 7- Does the diagram include all the interactions related to this functionality between the different objects? | X |  |
| 8- Does every object in the diagram that is shown to invoke a method of another object have access to perform that action? | X |  |
| 9- Does the diagram show the data returned from methods that have a return value? | X |  |

Minimal Class Diagram

|  |  |  |
| --- | --- | --- |
|  | YES | NO |
| 1- Do all classes have meaningful names? | X |  |
| 2- Do all classes have the right stereotype? | X |  |
| 3- Are all the associations between classes correct? | X |  |
| 4- Are all classes included in the diagram? | X |  |

### Static execution

This section includes the results obtained after walking through the artifacts produced in the analysis model.

Use Cases

|  |  |  |
| --- | --- | --- |
| **Unique ID** | **Test Result** | **Comments** |
| TS01 | Pass | It covers all the functionality in the TS013- RRComSys\_TextOnlyConnection use case. |
| TS02 | Pass | It covers all the functionality in the TS016-RRComSys\_TextAudioConnection use case. |
| TS03 | Pass | It covers all the functionality in the TS019 – RRComSys\_AudioVideoConnection use case. |
| TS04 | Fail | It covers all the functionality in the TS01- F03\_ReceiveFile  use case. |
| TS05 | Fail | It covers all the functionality in the TS04-F01\_SendFile use case. |
| TS06 | Fail | It covers all the functionality in the TS07-F08 – SecurityFileTransfer use case. |
| TS07 | Pass | It covers all the functionality in the TS010 – W02\_Workflow with Retry use case. |
| TS08 | Pass | It covers all the functionality in the TS022-W01\_WorkflowWithAlternateConnection use case. |

Sequence Diagrams

|  |  |  |
| --- | --- | --- |
| **Unique ID** | **Test Result** | **Comments** |
| TS\_01 | Pass | Covers all the sequence of events in the Text Connection Scenario sequence diagram. |
| TS\_02 | Pass | Covers all the sequence of events in the Text/Audio Connection sequence diagram. |
| TS\_03 | Pass | Covers all the sequence of events in the Audio/Video Connection Scenario sequence diagram. |
| TS\_04 | Pass | Covers all the sequence of events in the Receive File Scenario sequence diagram. |
| TS\_05 | Pass | Covers all the sequence of events in the Send File Scenario sequence diagram. |
| TS\_06 | Fail | Covers all the sequence of events in the Secure File Transfer Scenario sequence diagram. |
| TS\_07 | Pass | Covers all the sequence of events in the Workflow with Alternate Scenario sequence diagram. |
| TS\_08 | Pass | Covers all the sequence of events in the Workflow with Retry Scenario sequence diagram. |

Minimal Class Diagram

|  |  |  |
| --- | --- | --- |
| **Unique ID** | **Test Result** | **Comments** |
| TS\_01 | Pass | Covers all the classes and their associations in the CVM tool. |
| TS\_02 | Pass | Covers all the classes and their associations in the CVM tool. |
| TS\_03 | Pass | Covers all the classes and their associations in the CVM tool. |
| TS\_04 | Pass | Covers all the classes and their associations in the CVM tool. |
| TS\_05 | Pass | Covers all the classes and their associations in the CVM tool. |
| TS\_06 | Pass | Covers all the classes and their associations in the CVM tool. |
| TS\_07 | Pass | Covers all the classes and their associations in the CVM tool. |
| TS\_08 | Pass | Covers all the classes and their associations in the CVM tool. |

# Proposed Software Architecture

The RRComSSys will be broken down into several subsystems. These subsystems will be represented in a package diagram, according to two architectural patterns that were chosen with proper justification. Then the metamodel for the domain specific language will be specified. Also, the architecture will be represented by some UML profiles, which will consequently lead to the transformations expected from the architecture to the platform. Finally, the subsystems will be explained.

## Overview

The following figure presents the package diagram of the proposed system:



**Fig 4.1** RRComSSys Package diagram

**Description**: This is the package diagram for RRComSSys. It contains 6 subsystems. At first glance it can be seen that the pattern we are using is the Repository Architecture pattern. This pattern is used for maintaining the complex central body of information and it can be manipulated in wide variety of ways. In the project repository pattern holds the metadata for the models as well as the rule for our modeling languages.

## Meta-model for the DSL

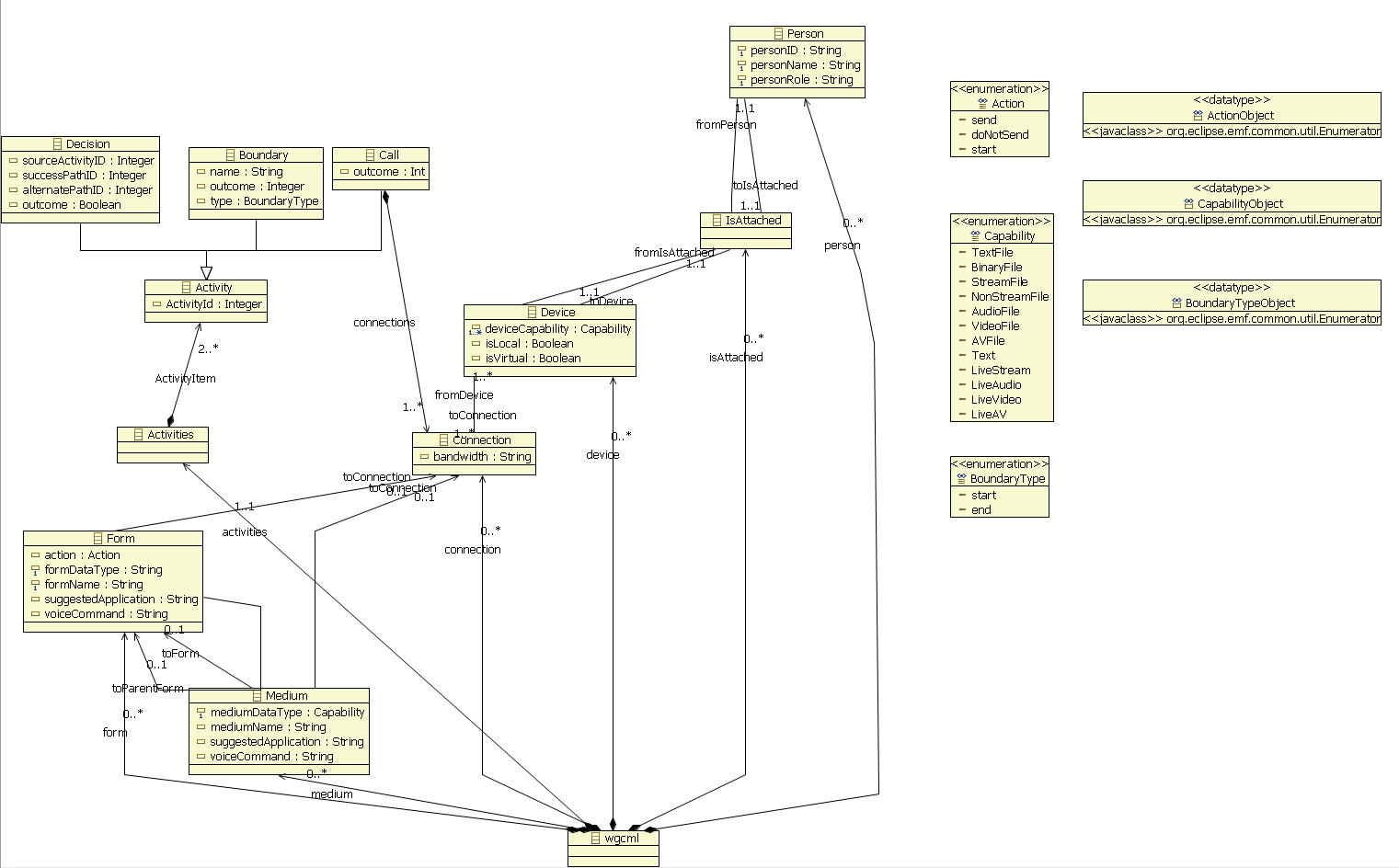


Figure 4.2.1 Class diagram for the Meta-model

**Description:** This class diagram represents the Meta-model used to create the model in the Communication Modeling Environment. It defines the grammar for model creation.

## UML profiles to represent the architecture

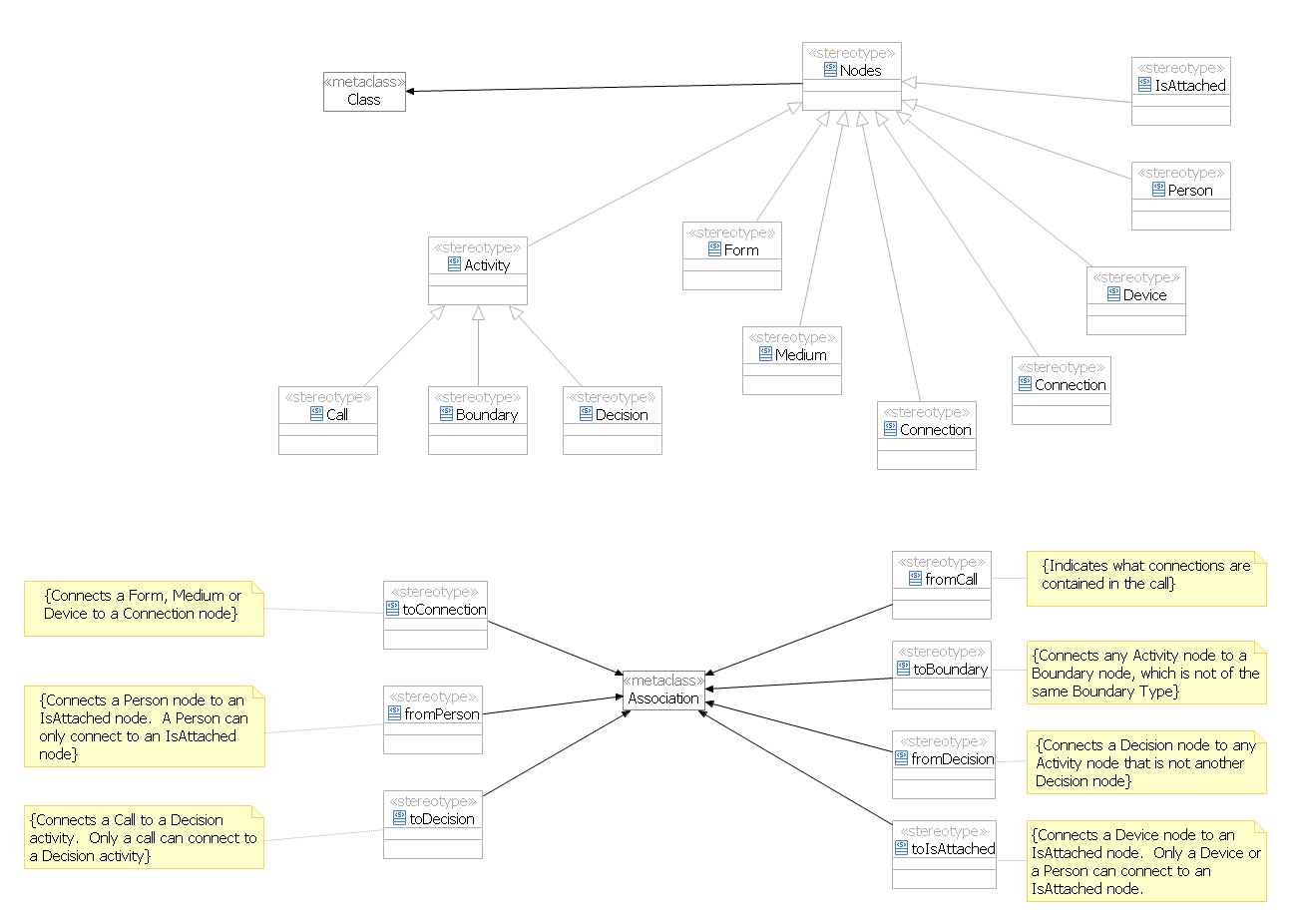


Figure 4.3.1 UML Profile for Model Construction

**Description:** This UML profile defines constraints when constructing a model.

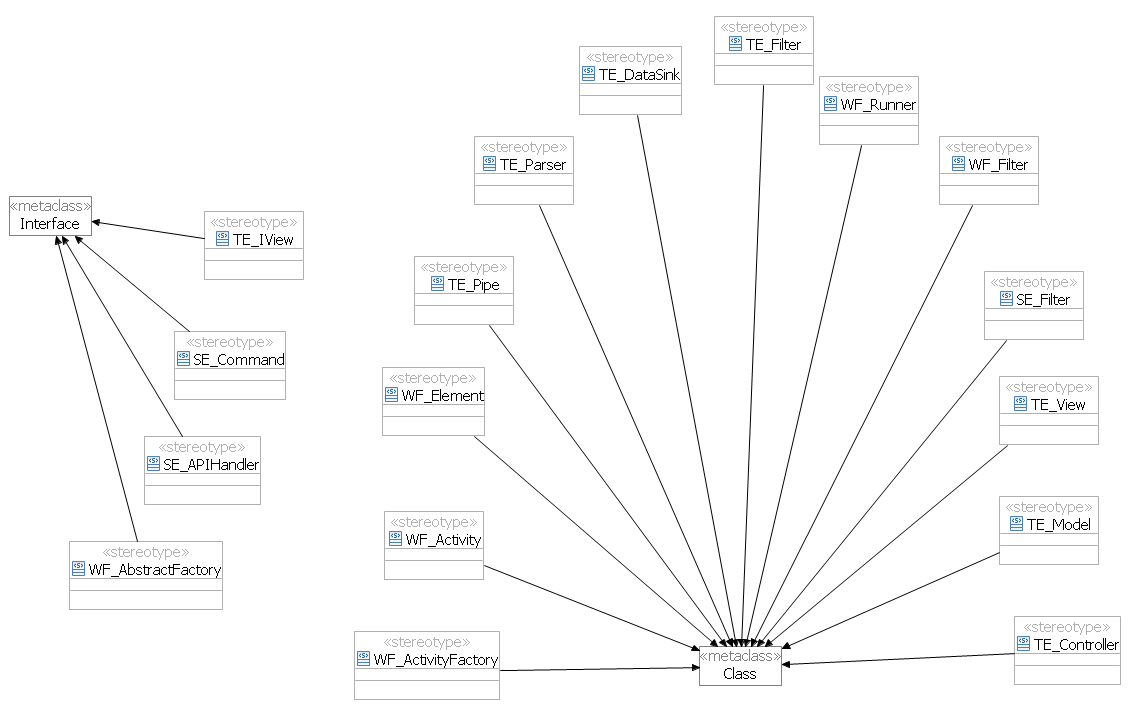


Figure 4.3.2 UML Profile for the Execution Architecture

**Description:** This UML Profile represents the architecture constraints on the components for the execution of a model.

## Generative Architecture

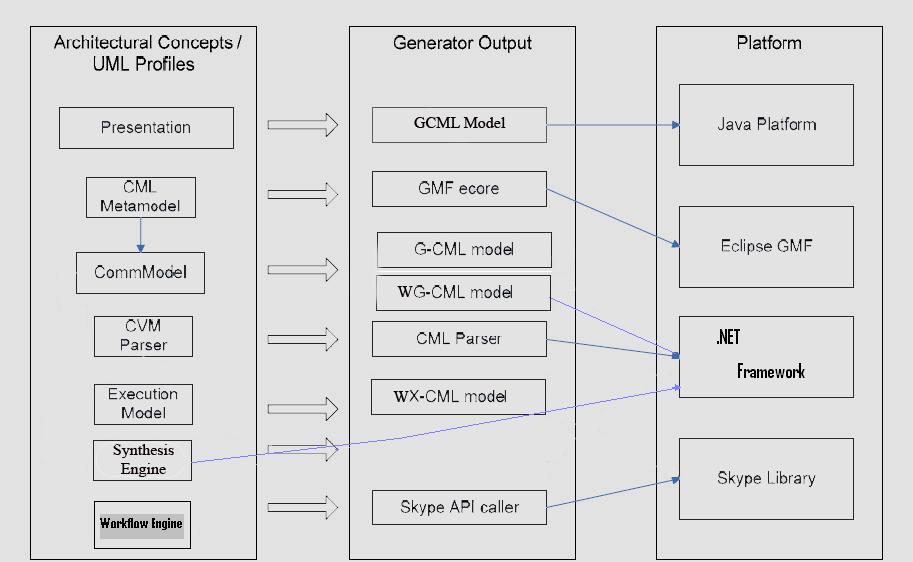


Figure 4.4.1 Generative Architecture Diagram

This section shows the transformation from UML concepts expressed in the various profiles to the target platform. As described, the presentation of a model is translated into a GCML model that is instantiable as a Java Object, or when transformed, as an equivalent object with the .NET Framework as a CLR (Common Language Runtime) object. The WCML Meta-model describe the semantics of a Communication Model that support workflow constructs, and these are translated into equivalent graphical schema of communication or hybrid communication/workflow models, which yet again are transformed into instantiable CLR objects. The Synthesis and Workflow Engines are concepts that describe components which are realized as .NET CLR assemblies which will process and execute the models built utilizing the Eclipse GMF. Skype API caller is a essentially a pass-through library for the interfaces the Synthesis and Workflow Engines and that invokes the functions provided by the Skype API. Execution Model is another UML profile concept that represents an Instance or a Schema model and gets transformed into a .wx-cml file; this file gets saved in the target platform OS file system.

## Subsystem Decomposition – describe each of the major subsystems

Communication User Interface – This is one of the main subsystems in the diagram. This subsystem contains all of the subsystems required to handle the creation of the user communication models. It also is responsible for the parsing and transformation of models. The workflow control is also performed here.

Modeling Environment – This subsystem deals with all that is the related to the modeling environment. This subsystem represents in the case of this project the Eclipse GMF and EMF. This subsystem is inherently Model-View-Control (MVC) as it inherits this from Eclipse itself.

Transformation Engine – This subsystem contains all the modules and classes related to the transformation of schemas into instances. This subsystem handles the conversion GXML into XCML and WXCML that will then be interpreted for execution.

Workflow Engine – This subsystem parses the WXCML data and creates the activity which can call or decides what should be done. It interacts with the synthesis engine to determine the order and process the activities.

Synthesis Engine – This subsystem hold all the classes required to transform XCML syntax into execution calls that can then be interpreted by a Communications Virtual Machine. In the case of this project this CVM is Skype.

Skype API – This subsystem represent our Communication Virtual Machine. This subsystem represents the Skype tool that we will be using in this project.

RRComSSys uses three architectural patterns justified below:

Model View Controller – This pattern is identified in the software architecture since it contains of three components such as model, view and controller. The model is used to contain the core functionality and the data. View obtains the data from the model. Controller receives the user input and translates the requests from the view or the model. This will later handled by the Pipe and Filter architecture. This pattern was chosen also because the target implementation platform GMF Eclipse is also Model View Controller. In conclusion, it comprises of the model, view and controller.

Pipe and Filter – This pattern provides a structure for the systems that process the stream of data. This pattern is essential for capturing the processing functionality needed for our system. Communication models have several stages and formats, so we push the input through a series of filters as the filter performs the function on the data and outputs the desired format. Filter also simplifies the implementation by having several stages in which the data will be in, making filter implementation and testing much easier. In conclusion, it comprises of a Pipe, Filter, Data Source, and Data Sink Subsystems.

Façade – This pattern uses an interface of a subsystem to facilitate the communication between subsystems.

## Validation of the System Model

This section describes the testing performed to determine if the system architecture and the metamodel satisfy the defined requirements.

### Checklist

The following checklist is used to inspect the completeness, correctness, and consistency of the system model.

Checklist for Architecture

|  |  |  |
| --- | --- | --- |
|  | YES | NO |
| 1- Do the subsystems exhibit high cohesion and low coupling? | X |  |
| 2- Do the architectural patterns selected make sense for the system? | X |  |
| 3- Are the associations between packages correct? | X |  |
| 4- Are all classes included in a package? | X |  |
| 5- Is the same naming scheme used for all packages? | X |  |
| 6- Was the same font style used throughout the diagram? | X |  |

Checklist for Meta-model

|  |  |  |
| --- | --- | --- |
|  | YES | NO |
| 1- Does the Meta-model diagram describe the syntax of the Communication Modeling Language? | X |  |
| 2- Does it define all constraints? |  | X |
| 3- Does it include all the possible constructs of the Communication Modeling Language? | X |  |
| 4- Are the relationships among the constructs of the language correct? | X |  |

### Static execution

This section includes the results obtained after walking through the artifacts produced in the system model.

Architecture

|  |  |  |
| --- | --- | --- |
| **Unique ID** | **Test Result** | **Comments** |
| TS01- F03\_ReceiveFile | Fail | It tests the functionality provided by the Transformation Engine Package and Synthesis Engine. |
| TS02- F03\_ReceiveFile | Fail | It tests the functionality provided by the Transformation Engine Package and Synthesis Engine. |
| TS03- F03\_ReceiveFile | Fail | It tests the functionality provided by the Transformation Engine Package and Synthesis Engine. |
| TS04- F01\_SendFile | Fail | It tests the functionality provided by the Transformation Engine Package and Synthesis Engine. |
| TS05- F01\_SendFile | Fail | It tests the functionality provided by the Transformation Engine Package and Synthesis Engine. |
| TS06- F01\_SendFile | Fail | It tests the functionality provided by the Transformation Engine Package and Synthesis Engine. |
| TS07- F08\_SecurityFileTransfer | Fail | It tests the functionality provided by the Transformation Engine Package. |
| TS08- F08\_SecurityFileTransfer | Fail | It tests the functionality provided by the Transformation Engine Package. |
| TS09- F08\_SecurityFileTransfer | Fail | It tests the functionality provided by the Transformation Engine Package. |
| TS010-W02\_Workflow with Retry | Pass | It tests the functionality provided by the Workflow Engine and Synthesis Engine. |
| TS011-W02\_Workflow with Retry | Pass | It tests the functionality provided by the Workflow Engine and Synthesis Engine. |
| TS012-W02\_Workflow with Retry | Pass | It tests the functionality provided by the Workflow Engine and Synthesis Engine. |
| TS013-RRComSys\_TextOnlyConnection | Pass | It tests the functionality provided by the Synthesis Engine and Transformation Engine. |
| TS014-RRComSys\_TextOnlyConnection | Pass | It tests the functionality provided by the Synthesis Engine and Transformation Engine. |
| TS015-RRComSys\_TextOnlyConnection | Pass | It tests the functionality provided by the Synthesis Engine and Transformation Engine. |

**Static Execution Continued**

|  |  |  |
| --- | --- | --- |
| TS016-RRComSSys\_TextAudioConnection | Pass | It tests the functionality provided by the Synthesis Engine and Transformation Engine. |
| TS017-RRComSSys\_TextAudioConnection | Pass | It tests the functionality provided by the Synthesis Engine and Transformation Engine. |
| TS018-RRComSSys\_TextAudioConnection | Pass | It tests the functionality provided by the Synthesis Engine and Transformation Engine. |
| TS019-RRComSSys \_AudioVideoConnection | Pass | It tests the functionality provided by the Synthesis Engine and Transformation Engine. |
| TS020-RRComSSys \_AudioVideoConnection | Pass | It tests the functionality provided by the Synthesis Engine and Transformation Engine. |
| TS021-RRComSSys \_AudioVideoConnection | Pass | It tests the functionality provided by the Synthesis Engine and Transformation Engine. |
| TS022-W01\_WorkflowWithAlternateConnection | Pass | It tests the functionality provided by the Workflow Engine and Synthesis Engine. |
| TS023-W01\_WorkflowWithAlternateConnection | Pass | It tests the functionality provided by the Workflow Engine and Synthesis Engine. |
| TS024-W01\_WorkflowWithAlternateConnection | Pass | It tests the functionality provided by the Workflow Engine and Synthesis Engine. |

# 5. Object Design

## 5.1 Overview – minimal class diagrams for the subsystems

This chapter will cover the Object Design process for the system by providing an overview of the minimal class diagram for the subsystems which are to be implemented. The sequence diagrams illustrate Object interactions. Furthermore, in the detailed class design the purpose of each class will be thoroughly explained.

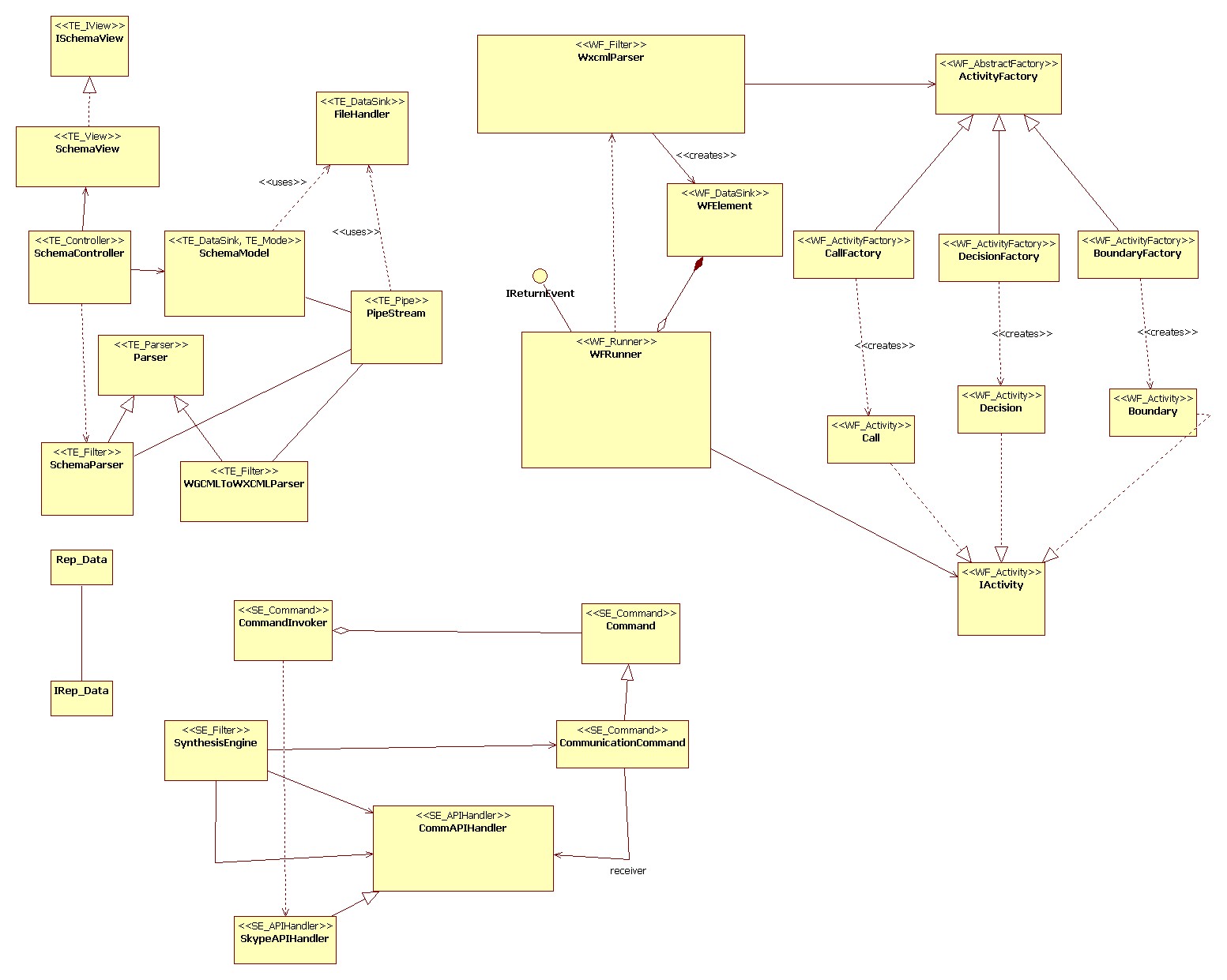


Figure 5.1.1 Minimal Class Diagram for RRComSSys

## 5.2 Object Interaction

**Object Diagram:**

UML object diagrams use a notation similar to class diagrams and are used to show an instance of a class at a particular point in time. You might want to draw an object diagram to illustrate a real-life example of a class and its relationships. UML object diagram can be a good option for explaining complex relationships between classes. The object diagrams are implemented for every scenario.

Refer to the Appendix C for the object diagrams.

# 

## 

## 

### Sequence Diagram

Sequence diagram depicts the sequence of actions that occur in a system. The invocation of methods in each object, and the order in which the invocation occurs is captured in a Sequence diagram. This makes the Sequence diagram a very useful tool to easily represent the dynamic behavior of a system.

Refer to the Appendix D for the sequence diagrams.

### State Machine

The State machine below describes the states and transitions during standard operation of the workflow engine within the CVM application.

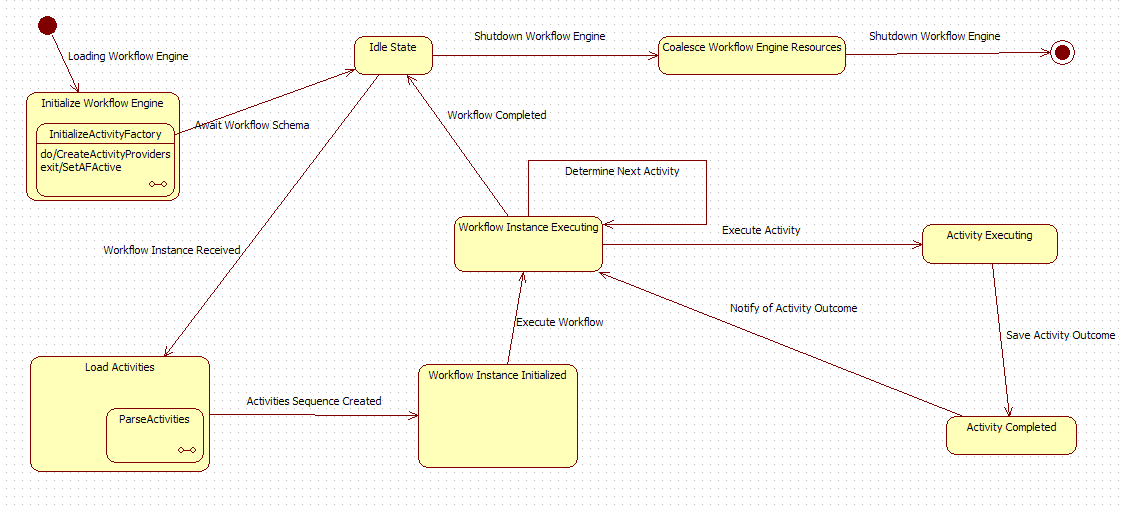


Figure 5.2.2.1 State Diagram for the Workflow engine within the CVM

## 5.3 Detailed Class Design

This section describes the purpose of each class in RRComSSys.

Refer to the detailed class diagram in appendix F

**Parser:** This abstract class extracts the similarities between the WGCMLToWXCMLParser and the SchemaParser. It dissects some xml source into an object model.

**WGCMLToWXCMLParser:** This class transforms the WGCML source to WXCML.

**SchemaParser:** This class validates the resulting WXCML schema and notifies the controller of any missing data in the schema.

**FileHandler:** This class handles any interaction with filesystem.

**PipeStream:** This class connects the output stream of a filter to the input stream of another filter or a datasink.

**SchemaModel:** This class holds the validated model and gets updated with any missing data by the SchemaController. Eventually this class will hold an instance of WXCML.

**SchemaController:** This class controls the SchemaView and updates the SchemaModel with new data.

**SchemaView:** This class presents the user with the forms necessary to get data missing from the schema.

**ISchemaView:** This interface extracts all relevant members and methods from the SchemaView. The SchemaController uses this data to control the SchemaView.

**WXCMLParser:** This class creates WFElements with the corresponding factory for a specific activity.

**WFRunner:** This class decides the order in which WFElements get processed.

**WFElement:** This class has an instance of the factory that will allow the WFRunner to create the appropriate Activity.

**IActivity:** This interface defines an Activity.

**Boundary:** This class represents a Boundary Activity. It also has logic to process a Boundary activity.

**Decision:** This class represents a Decision Activity. It also has logic to process a Decision activity.

**Call:** This class represents a Call Activity. It also has logic to process a Call activity.

**AbstractFactory:** This class encapsulates a group of factories that can create one of three Activities.

**CallFactory:** This class creates a Call Activity.

**BoundaryFactory:** This class creates a Boundary Activity.

**DesicionFactory:** This class creates a Decision Activity.

**IReturnEvent:** This interface defines the return event that will be triggered by the synthesis engine when a Call passes or fails.

**CommandInvoker:** This class asks the command to execute and in case of failure rollback.

**Command:** This class defines an interface for executing a CommunicationCommand.

**CommunicationCommand:** This class defines a binding between a CommAPIHandler object and an action. It also implements Execute by invoking the corresponding operations on Receiver.

**SynthesisEngine:** This class extracts and creates the commands out of an instance of a Call Activity.

**SkypeAPIHandler:** This is the receiver; it knows how to handle the commands through the Skype API.

## Validation of the system model

This section describes the testing performed to determine if the class diagrams and state chart diagram satisfy the defined requirements.

### Checklist

The following checklist is used to inspect the completeness, correctness, and consistency of the detailed design model.

Checklist for Detailed Class Diagram

|  |  |  |
| --- | --- | --- |
|  | YES | NO |
| 1- Do all attributes and operations have meaningful names? | X |  |
| 2- Do all classes have all their attributes? | X |  |
| 3- Is every attribute of the right type? | X |  |
| 4- Does every attribute have the right visibility? | X |  |
| 5- Do all classes have all their methods? | X |  |
| 6- Does every method have the right visibility? | X |  |
| 7- Do all associations show the right multiplicity? | X |  |
| 8- Are all patterns clearly defined? | X |  |

Checklist for State chart Diagram

|  |  |  |
| --- | --- | --- |
|  | YES | NO |
| 1- Does the diagram have an initial and final state? | X |  |
| 2- Does the diagram have transitions to connect every state? | X |  |
| 3- Does the diagram include actions, guards, and events? | X |  |
| 4- Are there any overlapping guards? |  | X |
| 5- Are all events correct? | X |  |
| 6- Are all guard conditions correct? | X |  |
| 7- Are all actions correct? | X |  |

### 5.4.2 Static Execution

This section includes the results obtained after walking through the artifacts produced in the detailed design model.

|  |  |  |
| --- | --- | --- |
| **Unique ID** | **Test Result** | **Comments** |
| TS01 | Pass | Covers all the classes including attributes and methods; and their associations in the development tool. |
| TS02 | Pass | Covers all the classes including attributes and methods; and their associations in the development tool. |
| TS03 | Pass | Covers all the classes including attributes and methods; and their associations in the development tool. |
| TS04 | Pass | Covers all the classes including attributes and methods; and their associations in the development tool. |
| TS05 | Pass | Covers all the classes including attributes and methods; and their associations in the development tool. |
| TS06 | Pass | Covers all the classes including attributes and methods; and their associations in the development tool. |
| TS07 | Pass | Covers all the classes including attributes and methods; and their associations in the execution tool. |
| TS08 | Pass | Covers all the classes including attributes and methods; and their associations in the execution tool. |

# 6. Implementation

This chapter deals with the realization of RRComSSys. Section 6.1 describes the software components needed for the system to run. While the following section, 6.2 describes the validation for the system; this is the most important validation as its’ primary expectation is to expose any and all the flaws that were not caught in the previous phases of development/testing.

## 6.1 Platform Specific Model (PSM)

There are several components that are needed to run RRComSSys (see Figure 6.1.1 below). In this implementation the Operating System platform is Microsoft Windows. WG-CML models are created using the Eclipse Modeling edition, which is specifically configured to facilitate the creation of model driven software artifacts via the interfaces and extensions provided by both EMF (Eclipse Modeling Framework), and GMF (Graphical Modeling Framework) on top of the standard Eclipse 3.4 IDE. The CVM, or platform specific components, is housed within a .NET WinForms application written in C#, which includes the Model transformation, workflow engine and which accesses the Skype API via a COM ( Component Object Model ) wrapper provided by Skype.

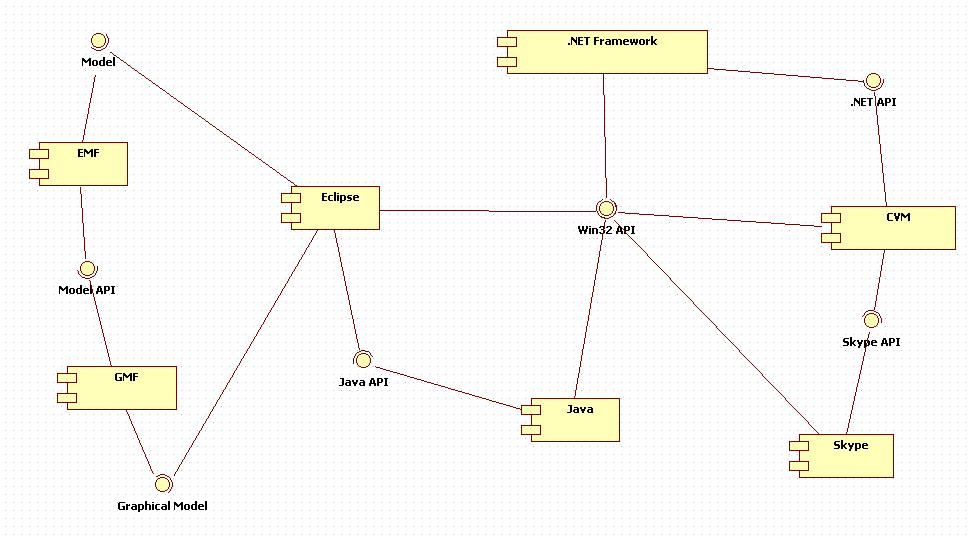


Figure 6.1.1 RRComSSys Platform Specific Component Model.

## 6.2 Validation of the Implementation

This section will cover the results of the test cases from section 3.5.1; by executing the test cases from the analysis model in the implementation the traceability and dependency between models as specified by USDP are preserved.

Test Cases Results

|  |  |  |
| --- | --- | --- |
| **Unique ID** | **Test Result** | **Comments** |
| TS01 | Pass |  |
| TS02 | Pass |  |
| TS03 | Pass |  |
| TS04 | Fail | Not implemented |
| TS05 | Fail | Not implemented |
| TS06 | Fail | Not implemented |
| TS07 | Pass |  |
| TS08 | Pass |  |

# 

# 7. Glossary

**Class Diagram** – A model representing the different classes within a software system and how they interact with each other.

**Component –** A physical and replaceable part of a system that conforms to and provides the realization of a set of interfaces.

**Model –** an abstract representation of a system that enables us to answer questions about the system.

**Post condition** – A predicate that must be true after an operation is invoked.

**Precondition –** A predicate that must be true before an operation is invoked.

**Sequence Diagram** – A model representing the different objects and/or subsystems of a software project and how they relate to each other during different operations for a given use case.

**Unified Modeling Language (UML)** – A standard set of notations for representing models.

**Use Case** – A general sequence of events that defines all possible actions between one or many actors and the system for a given piece of functionality.

**Model Driven Development Software –** Software development paradigm that roots in software product line engineering, which is the discipline of designing and building families of applications.

**Workflow Model** - A representation of actions, and activities that comprise a communication model that involve the usage of decision points that allow greater flexibility in describing models which can support more than one communication instance.

**Object Diagram** - A diagram that shows a complete or partial view of the structure of a modeled system or portion thereof.

**Shape** – An image that can be dragged into the canvas in the Visual Environment. It can be connected with other nodes using connections in order to build a communication model.

**Connections** – A line that connects one shape with another.

**Participant** – A special shape that represents a local or remote user of the communication system. One or more participants can establish a conversation using a communication model.

**Device** – A special shape that represents what the user is using to connect to the system.

**Canvas** – The area where the user drags shapes and connections to build a communication model.

**Capability** – The ways in which the device allows the user to communicate with other users.

**StartFlowBar** – Represents the starting point of a workflow model.

**EndFlowBar** – Represents the ending point of a workflow model.

**ConversationActivity** – A complete communication model that can be executed. If it fails, an alternate communication activity can be executed.

**DecisionPoint** – A special model component, denoted by a diamond shape, that dictates flow of a sequence of actions on the basis of the outcome of a previous action or activity.

# 8. Appendix

## 8.1 Appendix A – Use case diagram

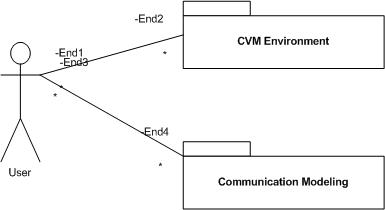


Figure 8.1.1 Use Case Diagram

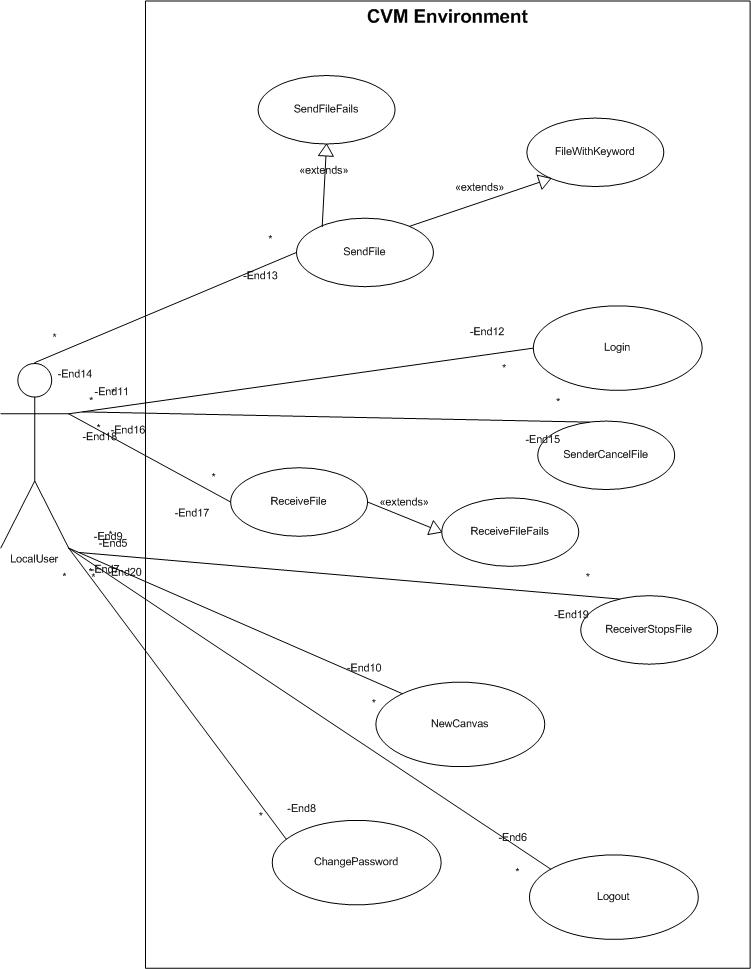


Figure 8.1.2 CVM Environment Use Case Diagram

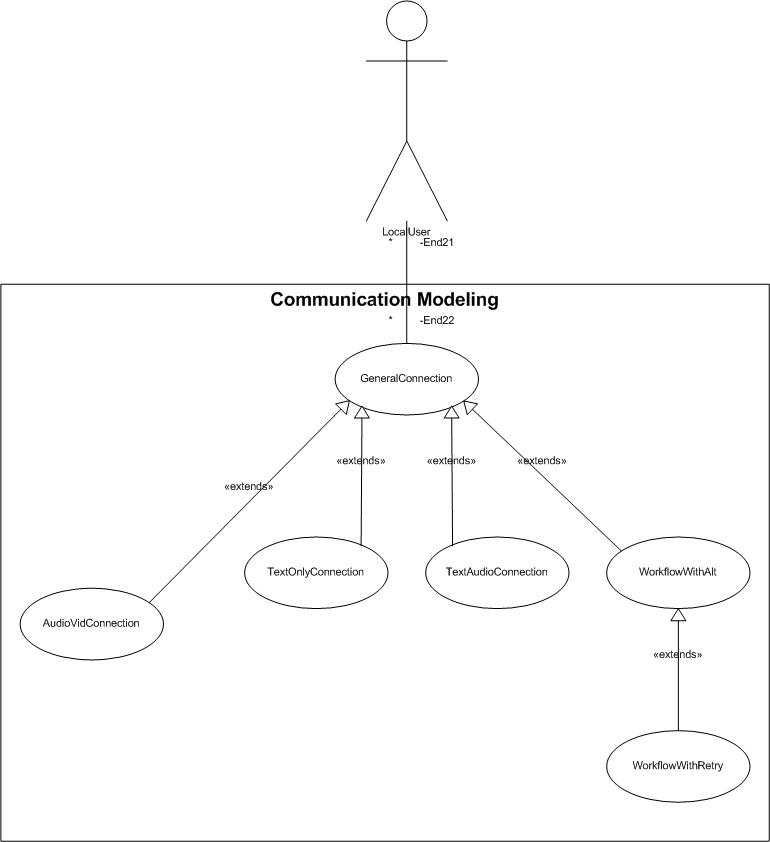


Figure 8.1.3 Communication Modeling Use Case Diagram

## 

## 8.2 Appendix B - Use Cases

### Use Case – Create a Text Only Connection

**Use Case ID:** RRComSSys\_TextOnlyConnection

**Use Case Level:** High-level

**Details:** Dr. Urbani wants to establish a text only connection with Dr. Salk.

* **Actor(s):** Dr. Urbani and Dr.Salk
* **Pre-conditions:** Dr. Urbani must have logged on (RRComSSys\_Login), a new file must be open (create a new file use case) and connection model has been set up using the procedure defined in use case RRComSSys\_GeneralConnection.
* **Description:**

**Trigger:**

**1.** Dr. Urbani drags the shape for text only capability onto the canvas. Dr. Urbani then attaches it to the “Device” node.

**2.** Dr. Salk must also have a device with the text only capability.

* **Post-conditions:** The canvas now holds a communication model that can be executed. When executed a connection must be created between Dr. Urbani and Dr. Salk and they should be able to chat using text.

**Alternative Courses of Action:** Dr. Urbani can add more participants using the same procedure to establish a multi-way connection.

**Extensions:** Depending on the “Capabilities” attached to the “Device” that the participants are using to create this model a Audio or Audio/Video connection (RRComSSys \_TextAudioConnection, RRComSSys \_AudioVideoConnection) can be establish when the communication model is executed. This connection can also be used to send/receive files.

**Exceptions:** The connection can’t be established.

**Concurrent Uses:** N/A

**Related Use Cases:** RRComSSys \_ GeneralConnection, RRComSSys \_TextAudioConnection, RRComSSys \_AudioVideoConnection, RRComSSys\_Login.

**---------------------------------------------------------------------------------------------------------------------**

**Decision Support**

**Frequency:** This use case will occur very often per application use. That is, every time that an actor wants to create communications models with chat support.

**Criticality:** High. It allows an actor to chat with another participant.

**Risk:** Low. There is little to no risk in this use case.

**Constraints:**

Each **participant** can only connect to (1) **IsAttached** shape. Each

**IsAttached** shape can only be connected to (1) **device** shape. Each **device** shape

must have at least (1) **capability** shape connected to it, and can only connect to (1) **connection** terminal. Each **capability** shape can only connect to (1) **device** shape.

* Usability: Easy to use. No need for technical knwoledge.
* Reliability: Connection must be established 90% of the time.
* Performance: Drawing a shape or connection shouldn’t take more than 1 second.
* Supportability: The device should have the latest Java Runtime Environment installed.
* Implementation: Implemented using Eclipse’s EMF and GMF plugins.

---------------------------------------------------------------------------------------------------------------------

**Modification History –** v1

**Owner:** Jean Rodriguez

**Initiation date:** 01/28/09

**Date last modified:** 01/29/09

### Use Case – Create a Text Audio Connection

**Use Case ID:** RRComSSys\_TextAudioConnection

**Use Case Level:** High-level

**Details:** Dr. Urbani wants to establish a Audio connection with Dr. Salk.

* **Actor(s):** Dr. Urbani and Dr.Salk
* **Pre-conditions:** Dr. Urbani must have logged on (Log in use case), a new file must be open (create a new file use case) and connection model has been set up using the procedure defined in use case RRComSSys\_GeneralConnection.
* **Description:**

**Trigger:**

**1.** Dr. Urbani drags the shape for audio capability onto the canvas. Dr. Urbani then attaches it to the “Device” node.

**2.** Dr. Salk must also have a device with the audio capability.

* **Post-conditions:** The canvas now holds a communication model that can be executed. When executed a connection must be created between Dr. Urbani and Dr. Salk and they should be able to speak with one another.
* **Alternative Courses of Action:** Dr. Urbani can add more participants using the same procedure to establish a multi-way connection.

**Extensions:** Depending on the “Capabilities” attached to the “Device” that the participants are using to create this model a Audio or Audio/Video connection (RRComSSys\_TextOnlyConnection, RRComSSys \_AudioVideoConnection) can be establish when the communication model is executed. This connection can also be used to send/receive files.

**Exceptions:** The connection can’t be established.

**Concurrent Uses:** N/A

**Related Use Cases:** RRComSSys \_ GeneralConnection, RRComSSys\_TextOnlyConnection, RRComSSys \_AudioVideoConnection, RRComSSys\_Login.

**---------------------------------------------------------------------------------------------------------------------**

**Decision Support**

**Frequency:** This use case will occur very often per application use. That is, every time that an actor wants to create communications models with audio chat support.

**Criticality:** High. It allows an actor to chat with another participant.

**Risk:** Low. There is little to no risk in this use case.

**Constraints:**

Each **participant** shape can only connect to (1) **IsAttached** shape. Each

**IsAttached** shape can only be connected to (1) **device** shape. Each **device** shape

must have at least (1) **capability** shape connected to it, and can only connect to (1) **connection** terminal. Each **capability** shape can only connect to (1) **device** shape.

* Usability: Easy to use. No need for technical knwoledge.
* Reliability: Connection must be established 90% of the time.
* Performance: Drawing a shape or connection shouldn’t take more than 1 second.
* Supportability: The device should have the latest Java Runtime Environment installed.
* Implementation: Implemented using Eclipse’s EMF and GMF plugins.

---------------------------------------------------------------------------------------------------------------------

**Modification History –** v1

**Owner:** Jean Rodriguez

**Initiation date:** 01/28/09

**Date last modified:** 01/29/09

### Use Case – Create a Audio Video Connection

**Use Case ID:** RRComSSys \_AudioVideoConnection

**Use Case Level:** High-level

**Details:** Dr. Urbani wants to establish a Audio/Video connection with Dr. Salk.

* **Actor(s):** Dr. Urbani and Dr.Salk
* **Pre-conditions:** Dr. Urbani must have logged on (Log in use case), a new file must be open (create a new file use case) and connection model has been set up using the procedure defined in use case RRComSSys\_GeneralConnection.
* **Description:**

**Trigger:**

**1.** Dr. Urbani drags the shape for audio/video capability onto the canvas. Dr. Urbani then attaches it to the “Device” node.

**2.** Dr. Salk must also have a device with the audio/video capability.

* **Post-conditions:** The canvas now holds a communication model that can be executed. When executed a connection must be created between Dr. Urbani and Dr. Salk and they should be able to speak and see one another.
* **Alternative Courses of Action:** Dr. Urbani can add more participants using the same procedure to establish a multi-way connection.

**Extensions:** Depending on the “Capabilities” attached to the “Device” that the participants are using to create this model a Audio or Text-only connection (RRComSSys\_TextOnlyConnection, C04\_AudioVideoConnection) can be establish when the communication model is executed. This connection can also be used to send/receive files.

**Exceptions:** The connection can’t be established.

**Concurrent Uses:** N/A

**Related Use Cases:** RRComSSys \_ GeneralConnection, RRComSSys\_TextOnlyConnection, RRComSSys \_AudioVideoConnection, RRComSSys\_Login.

**---------------------------------------------------------------------------------------------------------------------**

**Decision Support**

**Frequency:** This use case will occur very often per application use. That is, every time that an actor wants to create a communications model with audio/video support.

**Criticality:** High. It allows an actor to see and hear another participant.

**Risk:** Low. There is little to no risk in this use case.

**Constraints:**

Each **participant** shape must can only connect to (1) **IsAttached** shape. Each

**IsAttached** shape can only be connected to (1) **device** shape. Each **device** shape

must have at least (1) **capability** shape connected to it, and can only connect to (1) **connection** terminal. Each **capability** shape can only connect to (1) **device** shape.

* Usability: Easy to use. No need for technical knwoledge.
* Reliability: Connection must be established 90% of the time.
* Performance: Drawing a shape or connection shouldn’t take more than 1 second.
* Supportability: The device should have the latest Java Runtime Environment installed.
* Implementation: Implemented using Eclipse’s EMF and GMF plugins.

---------------------------------------------------------------------------------------------------------------------

**Modification History –** v1

**Owner:** Jean Rodriguez , Marcelo Lopez

**Initiation date:** 01/28/09

**Date last modified:** 01/29/09

### Use Case – Send File

**Use Case ID:** RRComSSys\_SendFile

**Use Case Level:** High

**Details:** This is a complete description of the use. Each subsection is explained below.

* **Actor(s):** Dr. Urbani and Dr. Salk
* **Pre-conditions:** A connection must be established(see Create a General Connection use case). Dr. Urbani must have rights to send the file. Dr. Urbani must have the send file capability. Dr. Urbani and Dr. Salk must be logged onto the system. Dr. Salk must accept the file.
* **Description:**

**Trigger:**

1. Dr. Urbani Executes the Select File use case.
2. Dr. Urbani than selects the file to be sent.

* **Relevant requirements:** The file is scanned if it is a compressed or executable file Dr. Salk recieves a warning of the potential danger.
* **Post-conditions:** The user receives a dialog conforming a succesful file transfer.

**Alternative Courses of Action** The file is too large. The file can not be sent. File transfer interupted. Dr. Salk or Dr. Salk exits application. Go to Error use case.

**Extensions:** Multiple files can be sent see use case F05\_RejectUnwantedFile.

**Exceptions:** File is too large. File can’t be read. File not found. Transfer failed. User disconected.

**Concurrent Uses:** N/A

**Related Use Cases:** RRComSSys\_SendFileFailed,RRComSSys\_ReceiveFile,RRComSSys\_ReceiveFileFailed, RRComSSys \_RejectUnwantedFile, RRComSSys\_SenderCancelsFile, RRComSSys\_FileTransferCancelled, RRComSSys\_SecurityFileTransfer

**------------------------------------------------------------------------------------------------------------------**

**Decision Support**

**Frequency:** This is mainly a messaging system so this will not be executed frequently. Approxamately 1 time pre execution.

**Criticality:** This function is only secondary towards live chat and live video.

**Risk:** High. This use case is key to providing this use.

**Constraints:**

* There will be a file size limit to a few MB. It must be reliable. Performance will not overwhelm system. Implemented on all systems that support file transfers. Not necessarily implemented on modile devices.

--------------------------------------------------------------------------------------------------------------

**Modification History**

**Owner:** Denis Antoine, Marcelo Lopez

**Initiation date:** 1/29/2009

**Date last modified:** 2/14/2009

### 

### Use Case – Receive File

**Use Case ID:** RRComSSys\_ReceiveFile

**Use Case Level:** High

**Details:**

* **Actor(s):** Dr. Salk and Dr. Urbani
* **Pre-conditions:** Receiver initiates the use case by accepting the file sent by Dr. Salk.
* **Description:**

**Trigger:**

1. Dr. Salk sends the file.

2. Dr. Salk accepts the file sent.

3. The file is than downloaded by Dr. Salk.

* **Relevant requirements:** Dr. Salk must be asked whether he or she wants the file.
* **Post-conditions:** The user receives a dialog conforming a succesful file transfer.

**Alternative Courses of Action** Dr. Salk does not accept the file.

**Extensions:** Multiple files can be received.

**Exceptions:** The file transfer is unsuccesful.

**Concurrent Uses:** Failed file transfers can occur. Files can be declined.

**Related Use Cases:** RRComSSys\_SendFile, RRComSSys\_SendFileFailed, RRComSSys\_ReceiveFileFailed, RRComSSys\_RejectUnwantedFile, RRComSSys \_SenderCancelsFile, RRComSSys\_FileTransferCancelled, RRComSSys\_SecurityFileTransfer

**---------------------------------------------------------------------------------------------------------------------**

**Decision Support**

**Frequency:** Always less than or equal to the frequency of RRComSSys\_SendFile.

**Criticality:** This function is only secondary towards live chat and live video.

**Risk:** High. All users must users must have the capability to accept file transfer requests.

**Constraints:**

* There will be a file size limit to a few MB. It must be reliable. Performance will not overwhelm system. Implemented on all systems that support file transfers. Not necessarily implemented on modile devices.

----------------------------------------------------------------------------------------------------------------

**Modification History**

**Owner:** Denis Antoine

**Initiation date:** 1/29/2009

Date last modified: 2/14/2009

### Use Case – Security File Transfer

**Use Case ID:** F08\_SecurityFileTransfer

**Use Case Level:** High

**Details:**

* **Actor(s):** Dr. Salk and Dr. Urbani
* **Pre-conditions:** A connection must be established(see Create a General Connection use case). Dr. Salk and Dr. Urbani must be logged on. Dr. Urbani has sent the file. Dr. Salk has accepted the file and a dialog asks Dr. Salk for an keyword
* **Description:**

**Trigger:**

1. Dr. Urbani initiates the action by executing the select secured file use case.
2. The system responds by prompting Dr. Urbani to enter a keyword than opens up a window to the desktop in which he selects the file.
3. Dr. Salk is than prompted to enter a keyword.
4. Upon a keyword match Dr. Salk is than prompted a file transfer request.

**Relevant requirements:** Dr. Salk and Dr. Urbani are notified of the initiated file transfer.

**Post-conditions:** The users receives a dialog conforming a succesful file transfer.

**Alternative Courses of Action** Dr. Salk or Dr. Urbani cancels the file sent. Dr. Salk enters an incorrect keyword.

**Extensions:**

**Exceptions:** Dr. Salk enters the correct keyword but it is incorrectly declined.

**Concurrent Uses: N/A**.

**Related Use Cases:** RRComSSys\_SendFile, RRComSSys\_SendFileFailed, RRComSSys\_ReceiveFile, RRComSSys\_ReceiveFileFailed, RRComSSys\_RejectUnwantedFile, RRComSSys \_SenderCancelsFile, RRComSSys\_FileTransferCancelled

**------------------------------------------------------------------------------------------------------------**

**Decision Support**

**Frequency:** Always less than or equal to the frequency of RRComSSys\_SendFile.

**Criticality:** Dr. Salk and Dr. Urbani must be able to send secure files to assure that the request is being received by a user who knows the keyword.

**Risk:** High. It allows only the intended recipetent to receive the sent file.

**Constraints:**

* It is simple to use with a mouse or via touch screen depending on the system type. It is supported on all system that can transfer files. It is implemented on all system that can transfer files.

-------------------------------------------------------------------------------------------------------

**Modification History**

**Owner:** Denis Antoine

**Initiation date:** 2/14/2009

Date last modified: 2/14/2009

### 

### Use Case – Workflow with Alternate Connection

**Use Case ID:** W01\_WorkflowWithAlternateConnection

**Use Case Level:** High-level

**Details:** Dr. Urbani wants to establish a connection with Dr. Salk, but if the connection is unsuccesful he wants to connect with Dr. Clarke.

* **Actor(s):** Dr. Urbani, Dr.Salk and Dr. Clarke
* **Pre-conditions:** Dr. Urbani must have logged on (RRComSSys\_Login), a new file must be open (RRComSSys\_CreateNewCanvas) and the canvas is set and ready to accept the new communication model.
* **Description:**

**Trigger:**

**1.** Dr. Urbani drags a “StartFlowBar” shape into the canvas.

**2.** Dr. Urbani sets up a general connection model following the procedure described in use case RRComSSys\_GeneralConnection. In this connection the other participant is Dr. Salk.

**3.** Dr. Urbani holds the left mouse button down and draws a rectangle that encapsulates the model created in step two. The connection is now a “ConversationActivity” shape.

**4.** Dr. Urbani connects the “ConversationActivity” shape created in step 3 with the “StartFlowBar” shape dragged in step 1.

**5.** Dr. Urbani drags a “DecisionPoint” shape into the canvas and connects it to the “ConversationActivity” created in step 3.

**4.** Repeats step 1-3 with Dr. Clarke as the other participant in the connection.

**5.** Dr. Urbani connects the “DecisionPoint” shape to the “ConversationActivity” created in step 4.

**6.** Dr. Urbani drags an “EndFlowBar” shape into the canvas.

**7.** Dr. Urbani connects both the “ConversationActivity”, created in step 5, and the “DecisionPoint” shape, to the “EndFlowBar” shape added in step 6.

* **Post-conditions:** The canvas now holds a communication model that can be executed. When executed a connection is attempted between Dr. Urbani and Dr. Salk. If that connection is successful, the connection is established. If that connection fails, the connection between Dr. Urbani and Dr. Clarke is executed.

**Alternative Courses of Action:** Dr. Urbani can add more participants using the same procedure to have multiple back up communication models. However, there must only be one start point and end point.

**Extensions:** Depending on the “Device” that the participants are using to create this model a Text only, Audio or Audio/Video connection (See RRComSSys\_TextOnlyConnection, RRComSSys\_TextAudioConnection, RRComSSys\_AudioVideoConnection) can be establish when the communication model is executed. This connection can also be used to send/receive files. The connections can be retried as in use case RRComSSys\_WorkflowWithRetry.

**Exceptions:** Both connections can’t be established.

**Concurrent Uses:** N/A

**Related Use Cases:** RRComSSys\_TextOnlyConnection, RRComSSys \_TextAudioConnection, RRComSSys\_AudioVideoConnection, RRComSSys\_Login, RRComSSys\_WorkflowWithRetry.

**---------------------------------------------------------------------------------------------------------------------**

**Decision Support**

**Frequency:** This use case will occur at least 50% of the time per application use. That is, every time that an actor wants to create a communications model to connect to another actor, with a backup connection in case that one fails.

**Criticality:** High. It allows an actor to create complex workflow connections with one or more participants.

**Risk:** High. The implementation is complex and some aspects are not clear.

**Constraints:**

Each **StartFlowBar** and **EndFlowBar** shape can only connect to one other **DecisionPoint** or **ConversationActivity** shape. Furthermore, there can only be one **StartFlowBar** and one **EndFlowBar** in the canvas.A **DecisionPoint** shape can only connect to a **ConversationActivity** or **EndFlowBar** shape. A **ConversationActivity** shape can connect to a **DecisionPoint** or **EndFlowBar** shape.

* Usability: Easy to use. No need for technical knwoledge.
* Reliability: Connection with one or the other participant must be established 90% of the time.
* Performance: Drawing a shape or connection shouldn’t take more than 1 second.
* Supportability: The device should have the latest Java Runtime Environment installed.
* Implementation: Implemented using Eclipse’s EMF and GMF plugins.

---------------------------------------------------------------------------------------------------------------------

**Modification History –** v1

**Owner:** Jean Rodriguez

**Initiation date:** 01/28/09

**Date last modified:** 01/29/09

### Use Case – Workflow with Retry

**Use Case ID:** RRComSSys\_Workflow with Retry

**Use Case Level:** High-level

**Details:** Dr. Urbani wants to establish a connection with Dr. Salk, but if the connection is unsuccesful Dr. Urbani wants to connect with Dr. Clarke. If both are unsuccsesful Dr. Urbani wants to try again.

* **Actor(s):** Dr. Urbani, Dr.Salk and Dr. Clarke
* **Pre-conditions:** Dr. Urbani must have logged on (RRComSSys\_Login), a new file must be open (RRComSSys\_CreateNewCanvas) and the canvas is set and ready to accept the new communication model.
* **Description:**

**Trigger:**

**1.** Dr. Urbani sets up a workflow conversation as described in use case W01\_WorkflowWithAlternateConnection.

**2.** Dr. Urbani connects the “DecisionPoint” shape to the “ConversationActivity” shape in which Dr. Salk is the other participant, as described in use case W01\_WorkflowWithAlternateConnection.

* **Post-conditions:** The canvas now holds a communication model that can be executed. When executed a connection is attempted between Dr. Urbani and Dr. Salk. If that connection is succesful, the connection is established. If that connection fails, the connection between Dr. Urbani and Dr. Clarke is executed. If the connection between Dr. Urbani and Dr. Clarke also fails, the connections are retried until one of the connections is succesful.

**Alternative Courses of Action:** Dr. Urbani can add more participants using the same procedure to have multiple back up communication models. However, there must only be one start point and end point.

**Extensions:** Depending on the “Device” that the participants are using to create this model a Text only, Audio or Audio/Video connection (See RRComSSys\_TextOnlyConnection, RRComSSys\_TextAudioConnection, RRComSSys\_AudioVideoConnection) can be establish when the communication model is executed. This connection can also be used to send/receive files.

**Exceptions:** Both connections can’t be established.

**Concurrent Uses:** N/A

**Related Use Cases:** RRComSSys\_TextOnlyConnection, RRComSSys\_TextAudioConnection, RRComSSys\_AudioVideoConnection, RRComSSys \_Login, RRComSSys\_WorkflowWithAlternateConnection.

**---------------------------------------------------------------------------------------------------------------------**

**Decision Support**

**Frequency:** This use case will occur at least 50% of the time per application use. That is, every time that an actor wants to create a communications model to connect to another actor, with a backup connection in case that one fails and with the ability to retry the connections.

**Criticality:** High. It allows an actor to create complex workflow connections with one or more participants and the ability to retry connections.

**Risk:** High. The implementation is complex and some aspects are not clear.

**Constraints:**

Each **StartFlowBar** and **EndFlowBar** shape can only connect to one other **DecisionPoint** or **ConversationActivity** shape. Furthermore, there can only be one **StartFlowBar** and one **EndFlowBar** in the canvas.A **DecisionPoint** shape can only connect to a **ConversationActivity** or **EndFlowBar** shape. A **ConversationActivity** shape can connect to a **DecisionPoint** or **EndFlowBar** shape.

* Usability: Easy to use. No need for technical knwoledge.
* Reliability: Connection with one or the other participant must be established 90% of the time.
* Performance: Drawing a shape or connection shouldn’t take more than 1 second.
* Supportability: The device should have the latest Java Runtime Environment installed.
* Implementation: Implemented using Eclipse’s EMF and GMF plugins.

---------------------------------------------------------------------------------------------------------------------

**Modification History –** v1

**Owner:** Jean Rodriguez

**Initiation date:** 01/28/09

**Date last modified:** 01/29/09

## 8.3 Appendix C – Object Diagram



**Figure 8.3.1:** **Object diagram for the Scenario-Text Only**



**Figure 8.3.2:** **Object diagram for the Scenario-Text and Audio**



**Figure 8.3.3:** **Object diagram for the Scenario-Text, Audio and video**



**Figure 8.3.4:** **Object diagram for the Scenario-Sending a File**



**Figure 8.3.5:** **Object diagram for the Scenario-Receive a File**



**Figure 8.3.6:** **Object diagram for the Scenario-Secured File Transfer**



**Figure 8.3.7:** **Object diagram for the Scenario-Work Flow Model**



**Figure 8.3.8:** **Object diagram for the Scenario- Ability to try until a connection is established**

# 

# 

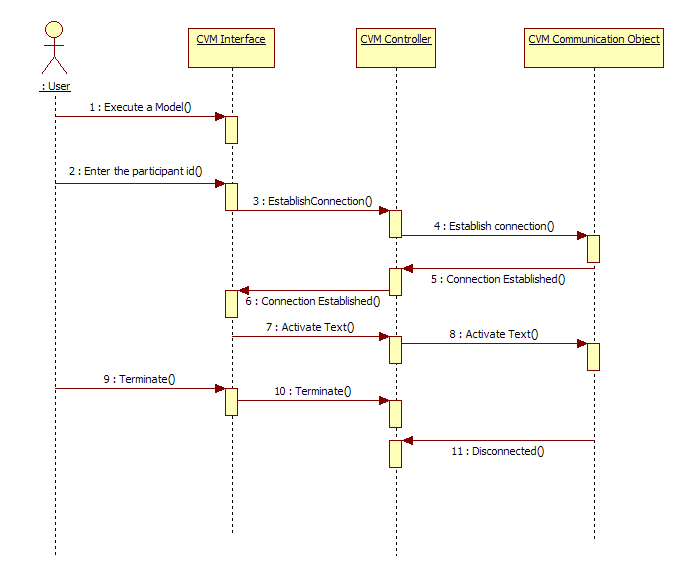
# 

## 

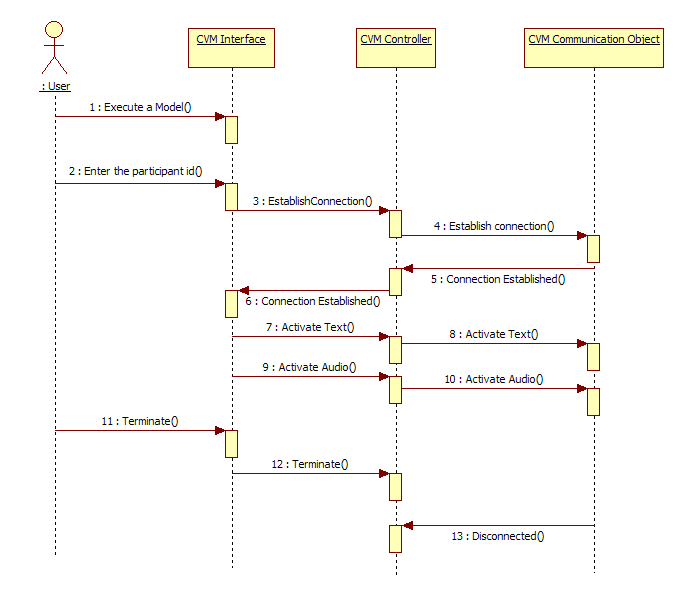
## 

## 

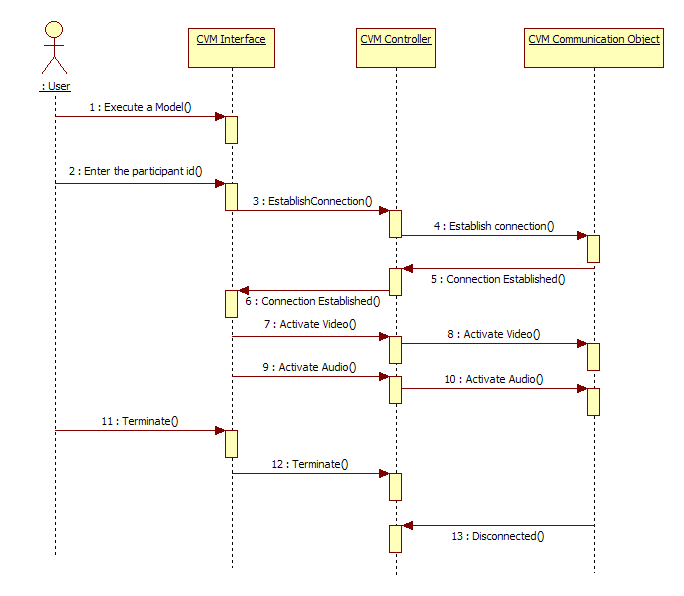
## Appendix D: Sequence diagrams



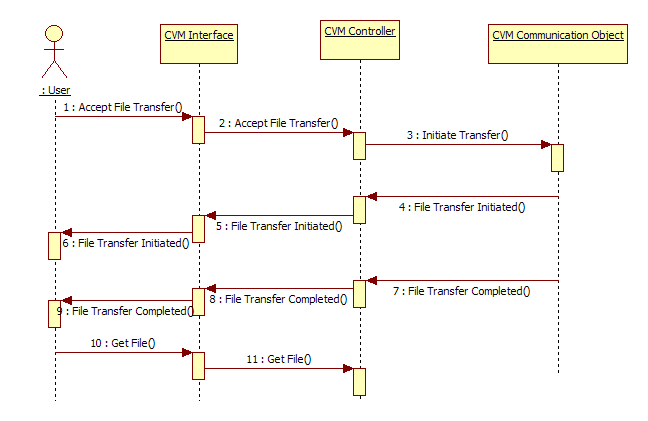
**Figure 8.4.1: Text Connection Scenario Sequence Diagram**



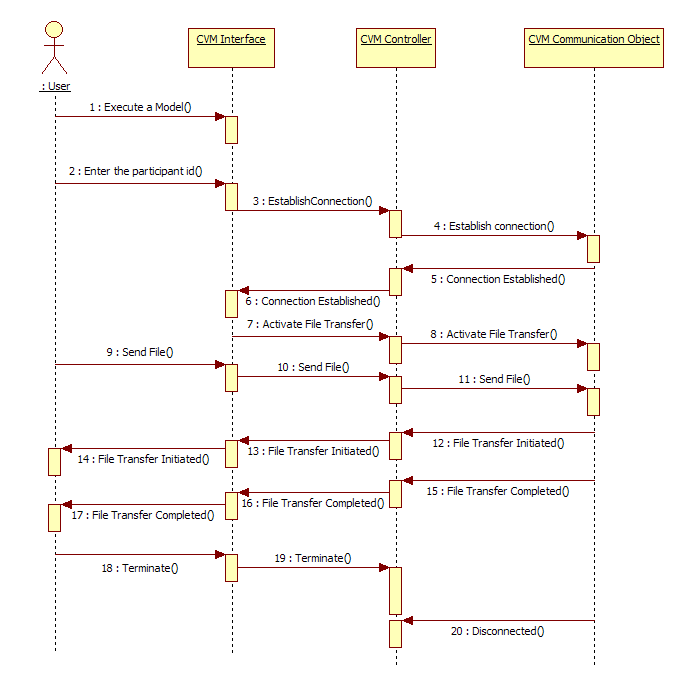
**Figure 8.4.2: TextAudio Connection Scenario Sequence Diagram**



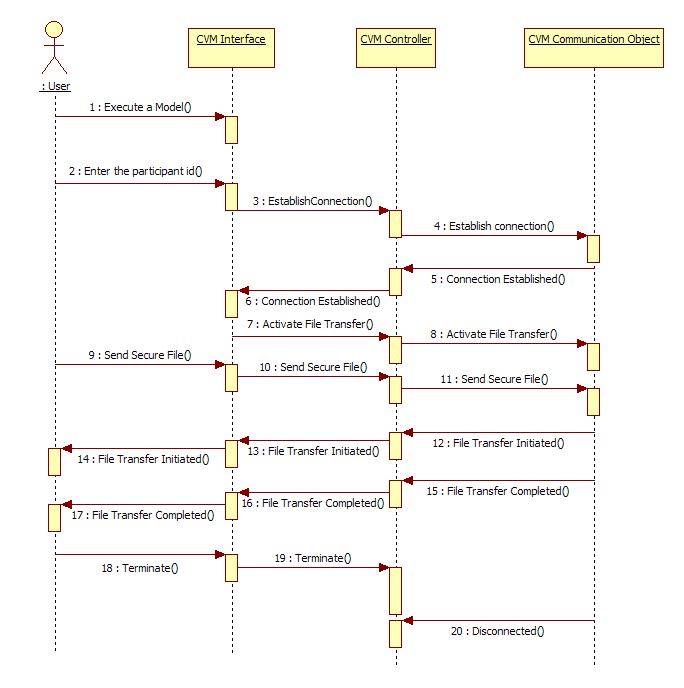
**Figure 8.4.3: Audio Video Scenario Sequence Diagram**



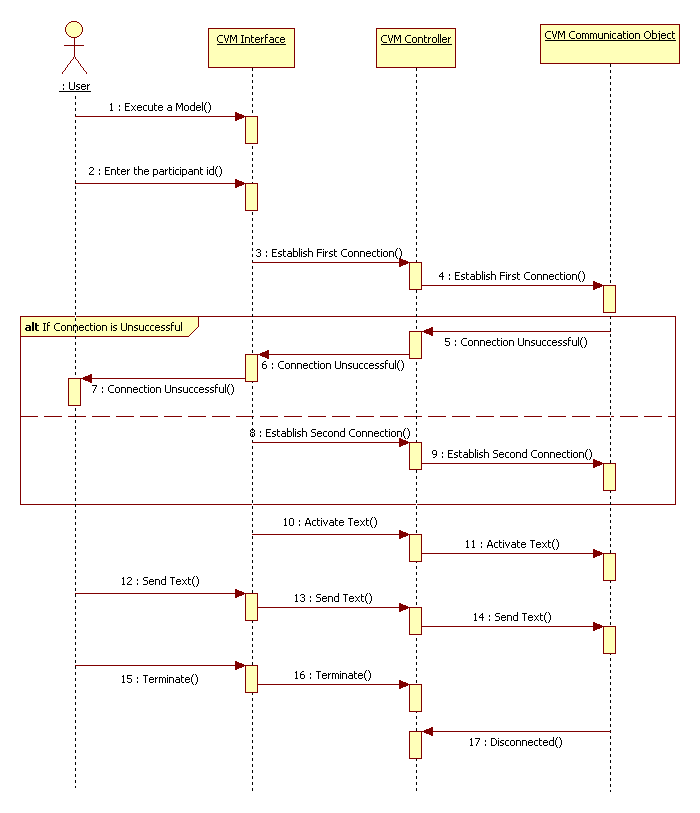
**Figure 8.4.4: Receive File Scenario Sequence Diagram**



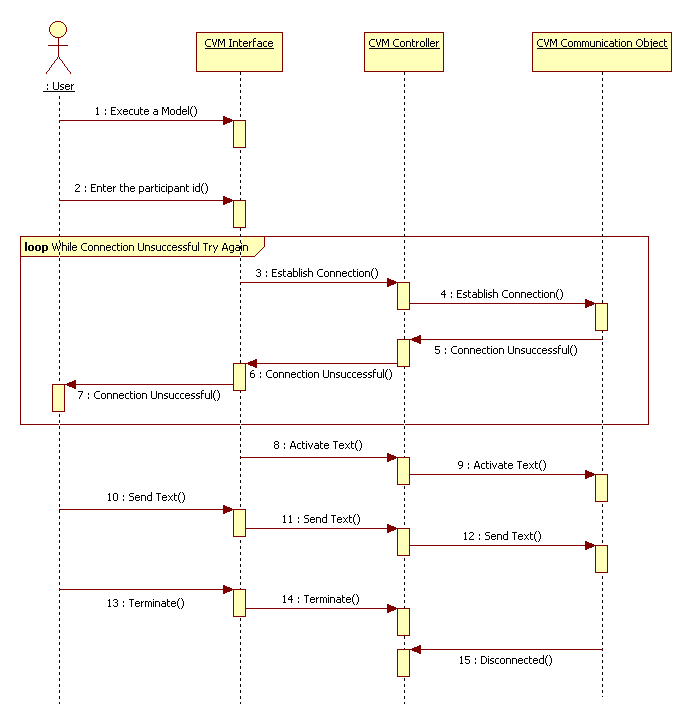
**Figure 8.4.5: Send File Scenario Sequence Diagram**



**Figure 8.4.6: Secure File Transfer Scenario with Keyword Sequence Diagram**

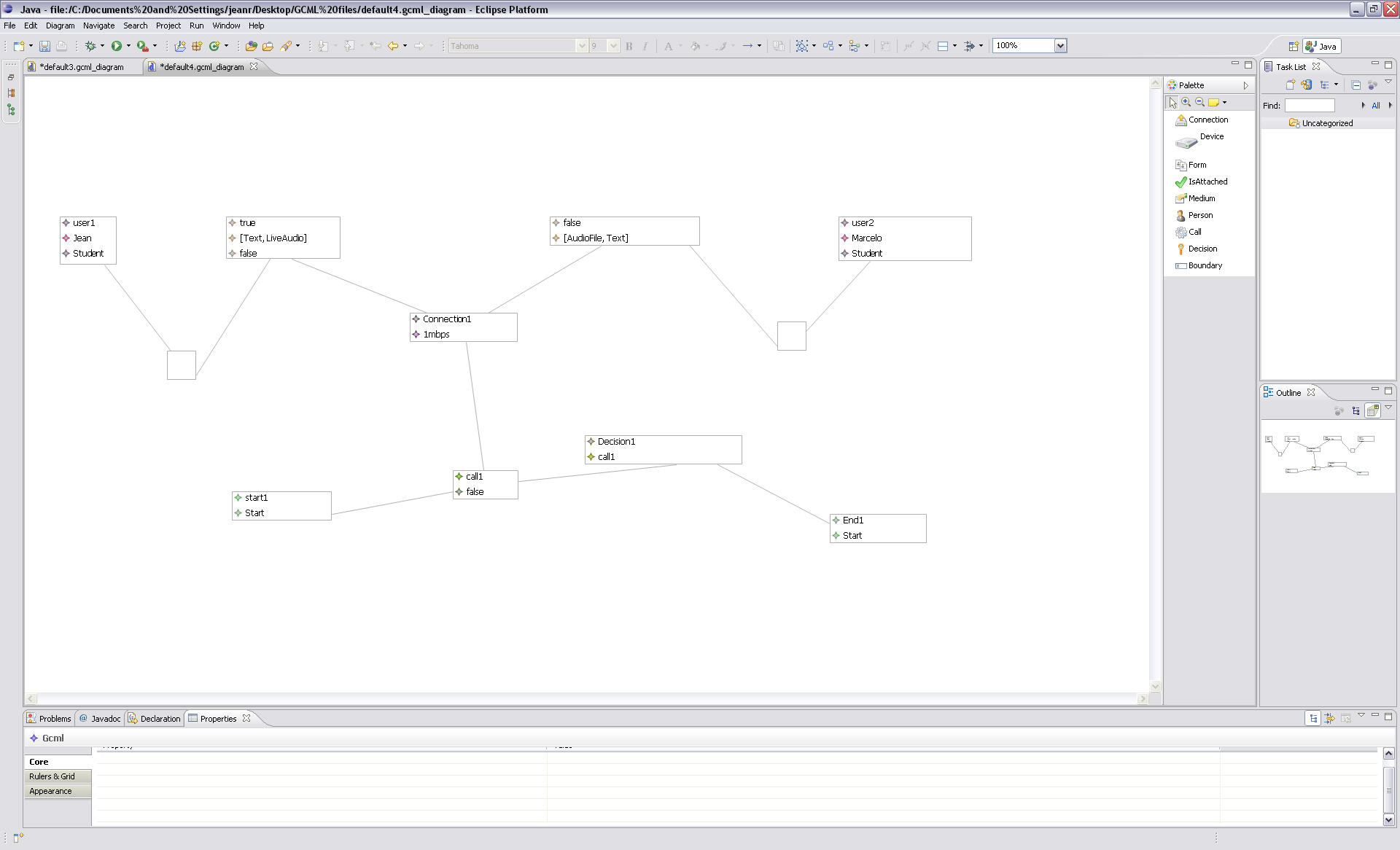


**Figure 8.4.7: Workflow with Alternate Scenario Sequence Diagram**

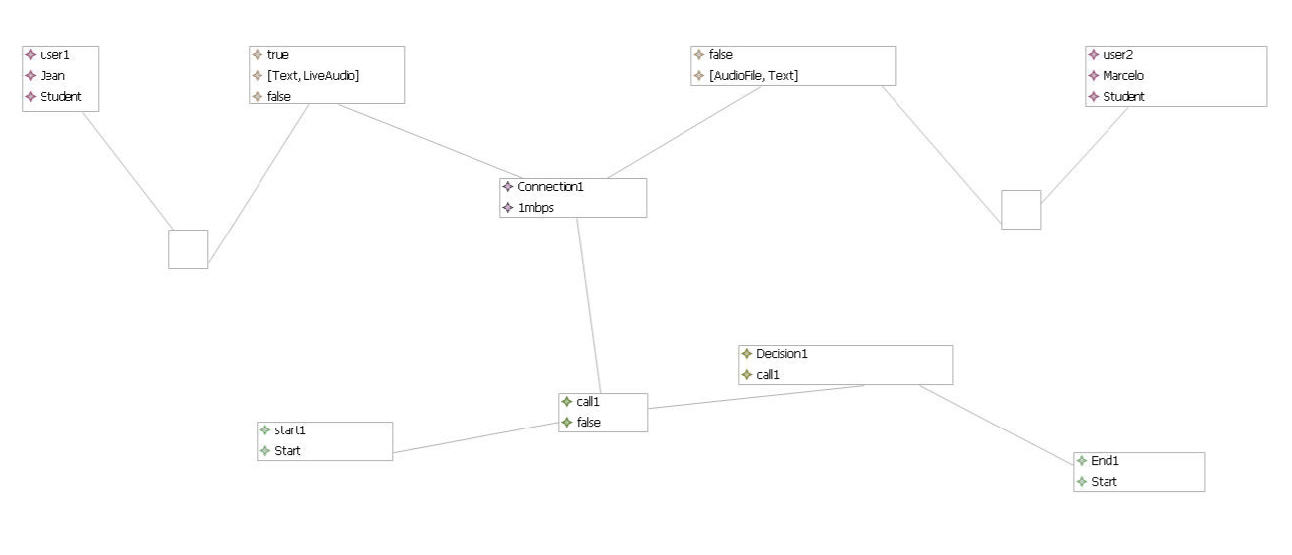


**Figure 8.4.8: Workflow with Retry Scenario**

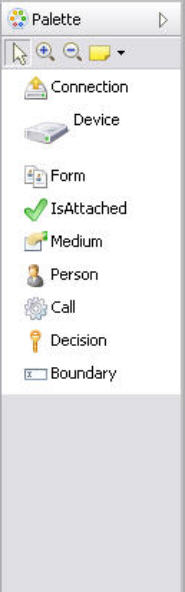
## Appendix E –User Interfaces



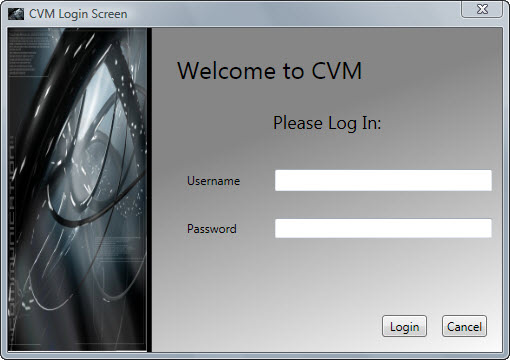
**Figure E1. Modeling Environment**



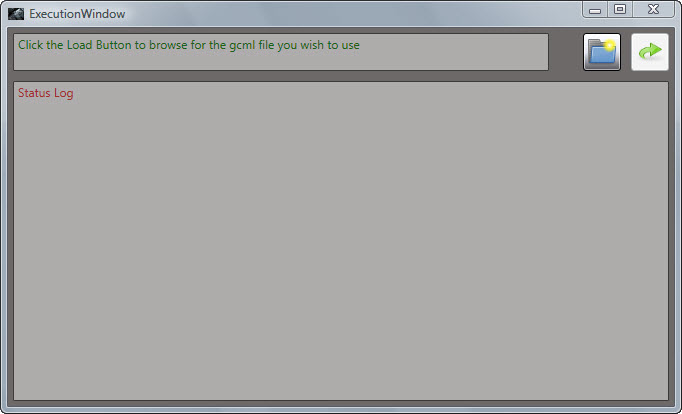
**Figure E2. Example Workflow Model**



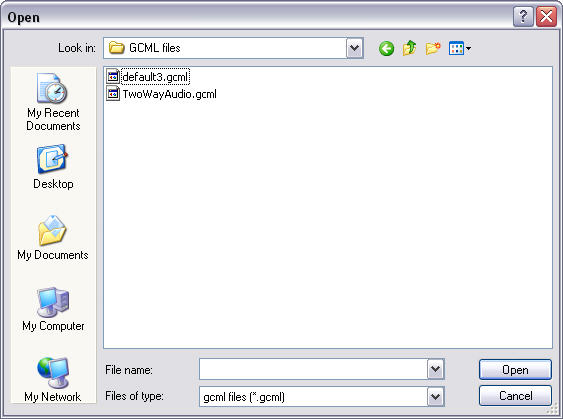
**Figure E3. Modeling Environment Palette**



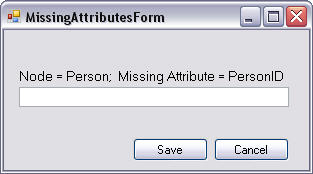
**Figure E4. Login Screen**



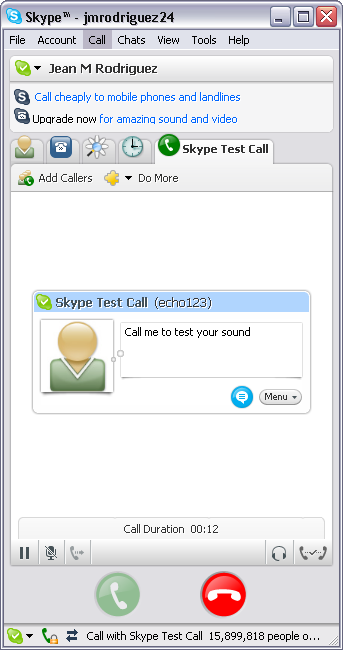
**Figure E5. File Loader**



**Figure E6. Choose File Dialog**



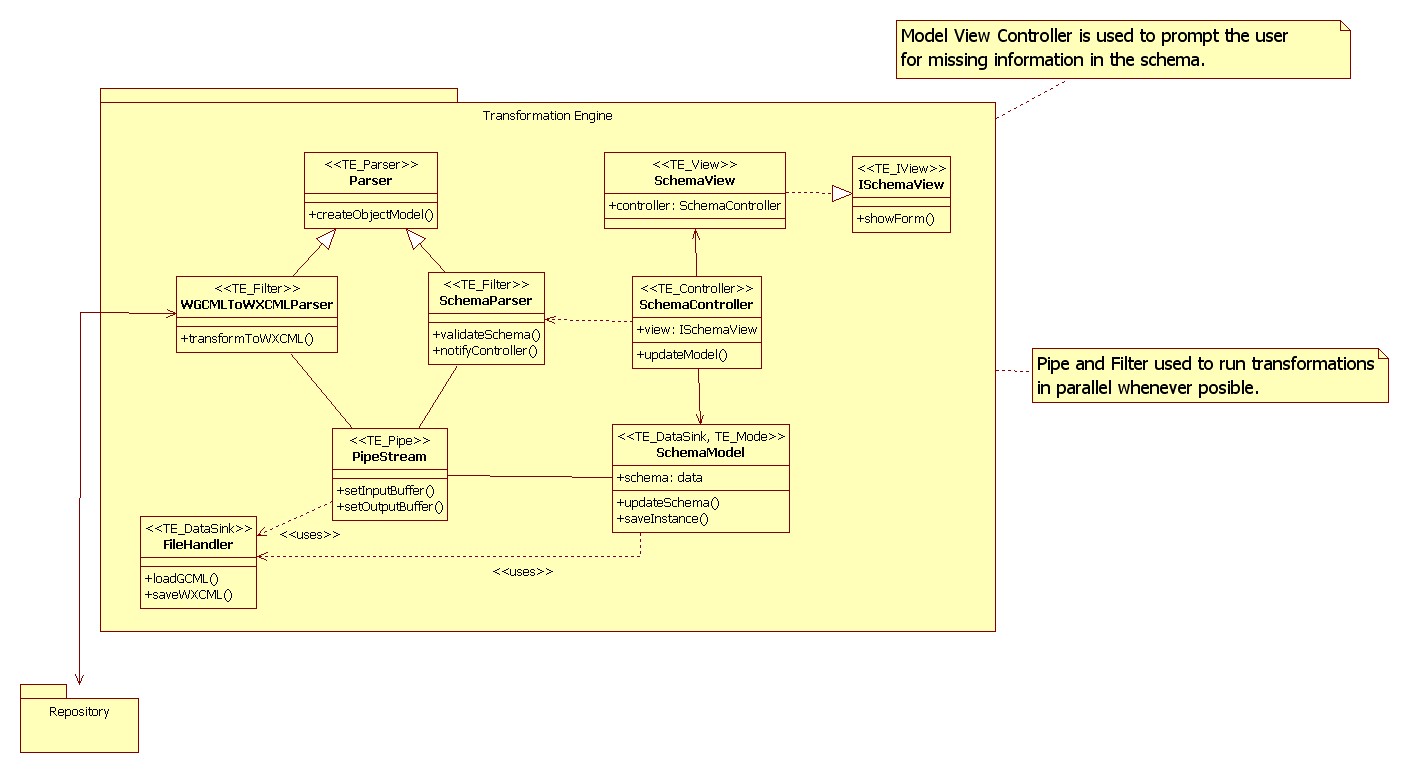
**Figure E7. Missing Attribute Form**



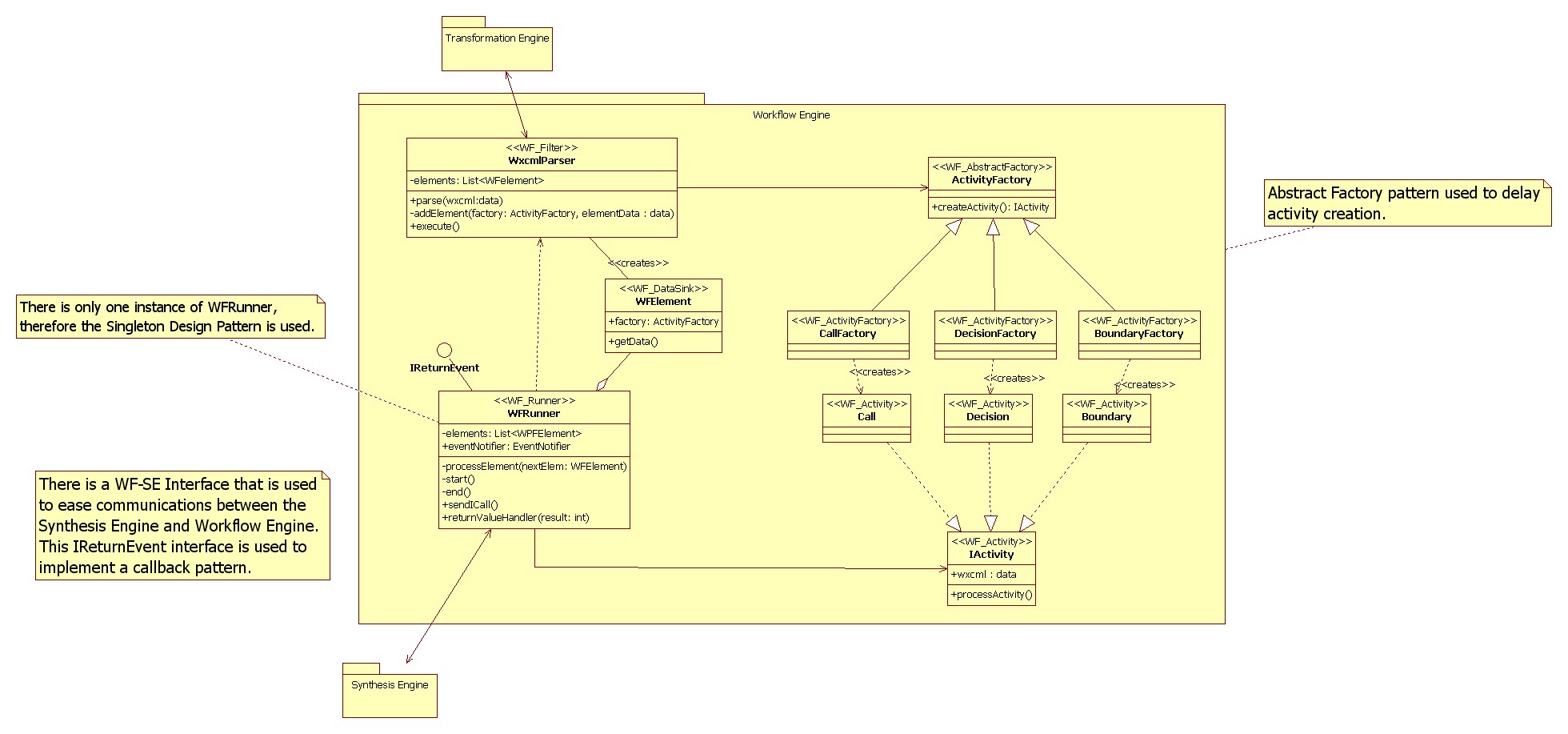
**Figure E8. Skype**

## 

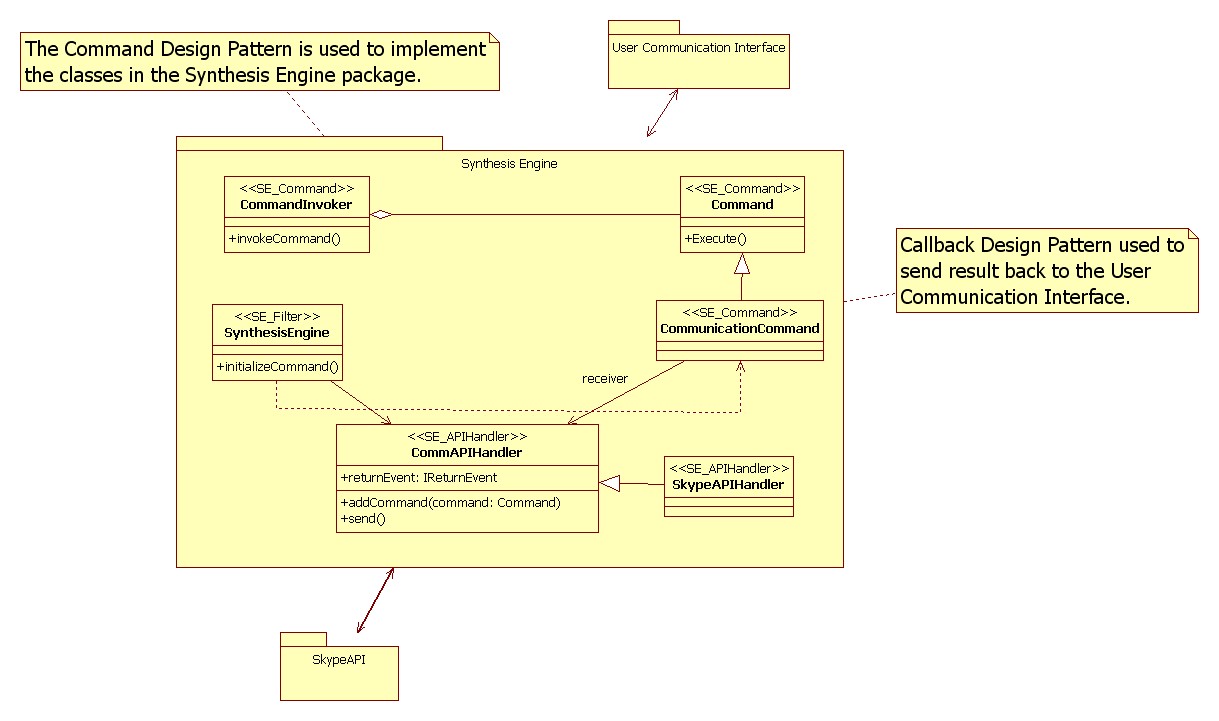
## 8.6 Appendix F – Detailed Class Diagram



**Figure 8.6.1 Detailed Class Diagram for the Transformation Engine**



**Figure 8.6.2 Detailed Class Diagram for the Workflow Engine**



**Figure 8.6.3 Detailed Class Diagram for the Synthesis Engine**



## Appendix G – Class Interfaces

//

//

// Generated by StarUML(tm) Java Add-In

//

// @ Project : Design Document

// @ File Name : ActivityFactory.java

// @ Date : 3/24/2009

// @ Author : RRComSSys Team five

//

//

/\*This class encapsulates a group of factories

\* that can create one of three Activities.\*/

public class ActivityFactory {

public IActivity createActivity() {

}

}

/\*This class represents a Boundary Activity.

\* It also has logic to process a Boundary activity.\*/

public class Boundary implements IActivity {

}

/\*This class creates a Boundary Activity.\*/

public class BoundaryFactory extends ActivityFactory {

}

/\*This class represents a Call Activity.

\* It also has logic to process a Call activity.\*/

public class Call extends IActivity {

}

/\*This class represents a Call Activity.

\* It also has logic to process a Call activity.\*/

public class Call extends IActivity {

}

/\*This class creates a Call Activity. \*/

public class CallFactory extends ActivityFactory {

}

/\*This class defines an interface

\* for executing a CommunicationCommand.\*/

public class Command {

public void Execute() {

}

}

/\*This class asks the command to execute and in case of failure rollback.\*/

public class CommandInvoker {

public void invokeCommand() {

}

}

/\*This class defines a binding between a CommAPIHandler object and an action.

\* It also implements Execute by invoking the corresponding operations on Receiver.\*/

public class CommunicationCommand extends Command {

}

/\*This class represents a Decision Activity.

\* It also has logic to process a Decision activity.\*/

public class Decision implements IActivity {

}

/\*This class creates a Decision Activity.\*/

public class DecisionFactory extends ActivityFactory{

}

/\*This class handles any interaction with filesystem.\*/

public class FileHandler {

public void loadGCML() {

}

public void saveWXCML() {

}

}

/\*This interface defines an Activity.\*/

public class IActivity {

public Object wxcml : data;

public void processActivity() {

}

}

/\*This interface defines the return event that

\* will be triggered by the synthesis engine when a Call passes or fails.\*/

public interface IReturnEvent {

boolean ReturnEvent();

}

/\*This interface extracts all relevant members and methods

\* from the SchemaView. The SchemaController uses this data

\* to control the SchemaView.\*/

public class ISchemaView {

public void showForm() {

}

}

/\*This abstract class extracts the similarities between the WGCMLToWXCMLParser and the SchemaParser.

\* It dissects some xml source into an object model.\*/

public class Parser {

public void createObjectModel() {

}

}

/\*This class connects the output stream of a filter to

\* the input stream of another filter or a datasink.\*/

public class PipeStream {

public void setInputBuffer() {

}

public void setOutputBuffer() {

}

}

/\*This class controls the SchemaView and updates

\* the SchemaModel with new data.\*/

public class SchemaController {

public ISchemaView view;

public void updateModel() {

}

}

/\*This class holds the validated model and gets updated

\* with any missing data by the SchemaController.\*/

public class SchemaModel {

public data schema;

public void updateSchema() {

}

public void saveInstance() {

}

}

/\*This class validates the resulting WXCML schema and notifies

\* the controller of any missing data in the schema.\*/

public class SchemaParser extends Parser {

public void validateSchema() {

}

public void notifyController() {

}

}

/\*This class presents the user with the forms

\* necessary to get data missing from the schema.\*/

public class SchemaView implements ISchemaView {

public SchemaController controller;

}

/\*This class extracts and creates the commands

\* out of an instance of a Call Activity.\*/

public class SynthesisEngine {

public void initializeCommand() {

}

}

/\*This class has an instance of the factory that

\* will allow the WFRunner to create the appropriate Activity.\*/

public class WFElement {

public ActivityFactory factory;

public void getData() {

}

}

/\*This class decides the order in which WFElements get processed.\*/

public class WFRunner implements IReturnEvent{

private List<WPFElement> elements;

public EventNotifier eventNotifier;

private void processElement(WFElement nextElem) {

}

private void start() {

}

private void end() {

}

public void sendICall() {

}

public void returnValueHandler(int result) {

}

}

/\*This class transforms the WGCML source to WXCML. \*/

public class WGCMLToWXCMLParser extends Parser {

public void transformToWXCML() {

}

}

/\*This class creates WFElements with the corresponding

\* factory for a specific activity.\*/

public class WxcmlParser {

private List<WFelement> elements;

public void parse(Object wxcml:data) {

}

private void addElement(ActivityFactory factory, Object elementData : data) {

}

public void execute() {

}

}

## 8.8 Appendix H - Project schedule



Figure 8.8.1 - Project Deliverable Schedule



Figure 8.8.1 - Project Plan Schedule Gantt Chart

## 8.9 Appendix I – Diary of meeting and tasks.

**Place:** Software Design Class

**Date:** 1/06/2009

**Start:** 8:30PM

**End:** 9:00PM

**Facilitator:** Jean Rodríguez

**Attending:** Jean, Denis,Marcelo,Barbara

**Minute Taker:**

**1. Objective**

Project kick off meeting.

**2.** **Status**

Everyone participated in coming up with ideas for the project.

**3. Discussion**

Discussion about how the GMF works.

**------------------------------------------------------------------------------------------------------------------**

**Place:** Software Design Class

**Date:** 1/13/2009

**Start:** 8:30PM

**End:** 9:00PM

**Facilitator:** Jean Rodríguez

**Attending:** Jean, Denis, Marcelo, Goutami

**Minute Taker:**

**1. Objective**

Project kick off meeting.

**2.** **Status**

Everyone participated in coming up with ideas for the project.

**3. Discussion**

Discussion about the Project.

**------------------------------------------------------------------------------------------------------------------**

**Place:** Graduate Lab

**Date:** 1/20/2009

**Start:** 8:30 PM

**End:** 9:30 PM

**Facilitator:** Jean Rodríguez

**Attending:** Jean, Denis, Harika, Marcelo

**Minute Taker:** Harika

**1. Objective**

Project kick off meeting.

**2.** **Status**

Everyone participated in coming up with ideas for the project.

**3. Discussion**

Give a brief explanation about the project to the new member.

**------------------------------------------------------------------------------------------------------------------**

**Place:** Graduate Lab

**Date:** 1/23/2009

**Start:** 4:00 PM

**End:** 4:30 PM

**Facilitator:** Jean Rodríguez

**Attending:** Jean, Denis, Harika

**Minute Taker:** Harika

**1. Objective**

Project kick off meeting.

**2.** **Status**

Everyone participated in coming up with ideas for the project.

**3. Discussion**

Discussion about how the GMF works.

**------------------------------------------------------------------------------------------------------------------**

**Place:** Graduate Lab

**Date:** 1/24/2009

**Start:** 11:00 AM

**End:** 2:00 PM

**Facilitator:** Jean Rodríguez

**Attending:** Jean, Denis, Harika, Marcelo

**Minute Taker:** Harika

**1.** **Objective**

Discuss on the Use Case Diagrams

**2.** **Status**

Use Case has been discussed and has an idea about the project.

**3. Discussion**

Saw a live demonstration of GMF usage. We discussed about the possible Use Cases for the project.

**4. Tasks**

All team members have to do the Use Case diagram given to them.

**------------------------------------------------------------------------------------------------------------------**

**Place:** Software Design Class

**Date:** 1/27/2009

**Start:** 8:30 PM

**End:** 9:00 PM

**Facilitator:** Jean Rodríguez

**Attending:** Jean, Denis, Harika, Marcelo

**Minute Taker:** Harika

**1.** **Objective**

Discuss on the presentation that should be given in next week.

**2.** **Status**

We discussed about the work to be done by each member.

**3. Discussion**

We discussed in detail about the presentation that will be given in next week.

**4. Tasks**

All team members have to do the work given to them.

**-----------------------------------------------------------------------------------------------------------------**

**Place:** Graduate Lab

**Date:** 1/30/2009

**Start:** 4:00 PM

**End:** 6:00 PM

**Facilitator:** Jean Rodríguez

**Attending:** Jean, Denis, Harika, Marcelo

**Minute Taker:** Harika

**1.** **Objective**

Discuss on the presentation that should be given in next week.

**2.** **Status**

We discussed about the work to be done by each member.

**3. Discussion**

We discussed in detail about the presentation that will be given in next week.

**4. Tasks**

All team members have to do the work given to them.

**------------------------------------------------------------------------------------------------------------------**

**Place:** Software Design Class

**Date:** 02/03/2009

**Start:** 8:30 PM

**End:** 9:15 PM

**Facilitator:** Jean Rodríguez

**Attending:** Jean, Denis, Harika, Marcelo

**Minute Taker:** Harika

**1. Objective**

Discuss about the First deliverable

**2.** **Status**

We have got a clear idea about the first deliverable.

**3. Discussion**

Work to be done for the deliverable.

**------------------------------------------------------------------------------------------------------------------**

**Place:** Graduate Lab

**Date:** 02/06/2009

**Start:** 6:00 PM

**End: 7**:00 PM

**Facilitator:** Jean Rodríguez

**Attending:** Jean, Denis, Harika, Marcelo

**Minute Taker:** Harika

**1. Objective**

Discuss about the First deliverable.

**2.** **Status**

Work is assigned that is to be done for the deliverable.

**3. Discussion**

General discussion about expectations.

**------------------------------------------------------------------------------------------------------------------**

**Place:** Software Design Class

**Date:** 02/10/2009

**Start:** 8:30 PM

**End:** 9:00 PM

**Facilitator:** Jean Rodríguez

**Attending:** Jean, Denis, Harika, Marcelo

**Minute Taker:** Harika

**1. Objective**

Discuss about the First deliverable presentation.

**2.** **Status**

Various tasks assigned to each member.

**3. Discussion**

The things to be done for the deliverable are discussed.

**------------------------------------------------------------------------------------------------------------------**

**Place:** Graduate Lab

**Date:** 02/14/2009

**Start:** 2:00 PM

**End:** 5:00 PM

**Facilitator:** Jean Rodríguez

**Attending:** Jean, Denis, Harika, Marcelo

**Minute Taker:** Harika

**1. Objective**

Discuss about the First deliverable.

**2.** **Status**

Work to be done for the deliverable such as the scenarios are discussed.

**3. Discussion**

We have discussed about the 8 most important use cases and writing the scenarios.

**------------------------------------------------------------------------------------------------------------------**

**Place:** Graduate Lab

**Date:** 02/15/2009

**Start:** 2:00 PM

**End:** 7:45 PM

**Facilitator:** Jean Rodríguez

**Attending:** Jean, Denis, Harika, Marcelo

**Minute Taker:** Harika

**1. Objective**

Discuss about the First deliverable. Collaborate to bring the SRAD together

**2.** **Status**

Work to be done for the deliverable such as the scenarios are discussed.

**3. Discussion**

We have discussed about the 8 most important use cases and writing the scenarios as well as the use cases, object, class. We also discussed the design of the work flow model.

**------------------------------------------------------------------------------------------------------------------**

**Place:** Graduate Lab

**Date:** 02/16/2009

**Start:** 7:00 PM

**End:** 11:00 PM

**Facilitator:** Jean Rodríguez

**Attending:** Jean, Denis, Harika, Marcelo

**Minute Taker:** Harika

**1. Objective**

Discuss about the First deliverable. Collaborate to bring the SRAD together

**2.** **Status**

Work to be done for the deliverable such as the scenarios are discussed.

**3. Discussion**

We have done all use cases, object and class diagrams. We also discussed the design of the work flow model.

**------------------------------------------------------------------------------------------------------------------**

**Place:** Graduate Lab

**Date:** 02/21/2009

**Start:** 8:00 PM

**End:** 9:00 PM

**Facilitator:** Denis Antoine

**Attending:** Jean, Denis, Harika, Marcelo

**Minute Taker:** Harika

**1. Objective**

Discuss the workflow.

**2.** **Status**

We have a clear idea of the workflow to be done.

**3. Discussion**

The workflow model has been discussed.

**------------------------------------------------------------------------------------------------------------------**

**Place:** Software Design Class

**Date:** 02/24/2009

**Start:** 8:30 PM

**End:** 9:00 PM

**Facilitator:** Denis Antoine

**Attending:** Jean, Denis, Harika, Marcelo

**Minute Taker:** Harika

**1. Objective**

Discuss about the Patterns and the work flow.

**2.** **Status**

Work to be done for the deliverable such as the workflow and the patterns are discussed.

**3. Discussion**

We have discussed the design of the work flow model. We have identified the patterns for our project.

**------------------------------------------------------------------------------------------------------------------**

**Place:** Graduate Lab

**Date:** 02/27/2009

**Start:** 6:00 PM

**End:** 7:30 PM

**Facilitator:** Denis Antoine

**Attending:** Jean, Denis, Harika

**Minute Taker:** Harika

**1. Objective**

Discuss about how to generate a GMF model.

**2.** **Status**

Tried to generate a GMF model from an sample example.

**3. Discussion**

We have tried how to create a Meta model in EMF, convert it to GMF and generate a final model.

**------------------------------------------------------------------------------------------------------------------**

**Place:** Software Design Class

**Date:** 03/03/2009

**Start:** 8:30 PM

**End:** 9:00 PM

**Facilitator:** Denis Antoine

**Attending:** Jean, Denis, Harika, Marcelo

**Minute Taker:** Harika

**1. Objective**

Discuss the Meta model and the Package diagram.

**2.** **Status**

We have discussed details about the Meta model such as the DSL.

**3. Discussion**

Talk about the Domain, Meta model, syntax, semantics, DSL.

**------------------------------------------------------------------------------------------------------------------**

**Place:** Graduate Lab

**Date:** 03/07/2009

**Start:** 2:00 PM

**End:** 6:30 PM

**Facilitator:** Denis Antoine

**Attending:** Jean, Denis, Harika, Marcelo

**Minute Taker:** Harika

**1. Objective**

Study for exam 2.

**2.** **Status**

We have discussed the exam by creating the DSL and the profile for the scenario of our project.

**3. Discussion**

We have created the DSL by considering syntax and semantics and also tried to draw a UML Profile by considering the scenario of our project.

**------------------------------------------------------------------------------------------------------------------**

**Place:** Software Design Class

**Date:** 03/10/2009

**Start:** 8:30 PM

**End:** 9:00 PM

**Facilitator:** Denis Antoine

**Attending:** Jean, Denis, Harika, Marcelo

**Minute Taker:** Harika

**1. Objective**

Split the sections off to the users.

**2.** **Status**

Work to be done for the deliverable

**3. Discussion**

Various sections assigned to all members.

**------------------------------------------------------------------------------------------------------------------**

**Place:** Graduate Lab

**Date:** 03/14/2009

**Start:** 2:00 PM

**End:** 7:45 PM

**Facilitator:** Denis Antoine

**Attending:** Jean, Denis, Marcelo, Harika

**Minute Taker:** Harika

**1. Objective**

Discuss the Object design.

**2.** **Status**

We tried to run the canvas.

**3. Discussion**

We tried to get the canvas running but it is partially running.

**------------------------------------------------------------------------------------------------------------------**

**Place:** Graduate Lab

**Date:** 03/22/2009

**Start:** 2:00 PM

**End:**  5:00PM

**Facilitator:** Denis Antoine

**Attending:** Jean, Denis, Marcelo

**Minute Taker:** Harika

**1. Objective**

Discuss various discrepancies.

**2.** **Status**

We have done with most of the part in the second deliverable.

**3. Discussion**

We have discussed about the state machine, Meta model. The remaining work that should be done is also assigned.

**------------------------------------------------------------------------------------------------------------------**

**Place:** Graduate Lab

**Date:** 03/22/2009

**Start:** 2:00 PM

**End:**  6:00PM

**Facilitator:** Denis Antoine

**Attending:** Jean, Denis, Marcelo

**Minute Taker:** Harika

**1. Objective**

Discuss the Second deliverable.

**2.** **Status**

Section 1, 4, and 5 are complete.

**3. Discussion**

Finalize the remaining sections.

**------------------------------------------------------------------------------------------------------------------**

**Place:** Graduate Lab

**Date:** 03/27/2009

**Start:** 2:00 PM

**End:**  5:00PM

**Facilitator:** Marcelo Lopez

**Attending:** Jean, Denis, Marcelo, Harika

**Minute Taker:** Harika

**1. Objective**

Discuss about the final project

**2.** **Status**

We discussed regarding the development of the code.

**3. Discussion**

We finalized the tasks to be completed.

**------------------------------------------------------------------------------------------------------------------**

**Place:** Graduate Lab

**Date:** 04/11/2009

**Start:** 2:00 PM

**End:**  3:00PM

**Facilitator:** Marcelo Lopez

**Attending:** Jean, Denis, Marcelo, Harika

**Minute Taker:** Harika

**1. Objective**

Discuss about the final project document and the presentation

**2.** **Status**

We assigned work for the final document.

**3. Discussion**

We finalized the tasks to be completed for the final presentation.

**------------------------------------------------------------------------------------------------------------------**