April 16, 2007
CEN 4021 - Software Engineering II/CEN 5064 - Software Design
Professor: Peter Clarke

Rapid Realization of Communication Services System

Team #1
Integrants:

1. Alejandro Ortiz
2. Ariel Cary
3. Frank Hernandez

Abstract

Today we encounter several types of communication applications and technologies: chat messaging, voice over IP, cellular telephony, etc. Typically, the development of a communication application that integrates these diverse technologies is a complex and costly process; especially a model that abstracts out the specific implementation platforms. This is despite of the fact that there currently exist visual environments and high-level programming languages. Recently, there has been some work done in this direction. In particular, [1] proposes a declarative language for specifying user-centric communication schemas that are network and device independent.

This document contains the requirements, project plan and system model documentations for a system based in [1], which we call "*Rapid Realization of Communication Services System*". This system will practically allow a user to create a communication model using an easy to use graphical infrastructure, which allows the user to export and transform this model in order to make phone calls and send chat messages. This system will also allow developers to implement any variety of communication models that they wish, simply by dragging shapes and connections into the modeling environment. In a matter of minutes he will have implemented a working model to establish any kind of communication between any kind and number of end users.

Table of Contents

1. Introduction

The following is an introduction to Rapid Realization of Communication Services System. In this chapter, the purpose and scope of the system, as well as any necessary term definitions, acronyms, and abbreviations shall be defined. Finally, an overview of this document will be outlined.

1.1 Purpose of System

In recent years, communications have been shifting dramatically from the phone lines towards the digital form. Chat messaging, video conferencing, voice over IP, and countless other forms of digital communication are taking over. It is cheaper, faster, and increasingly more reliable as technology improves. Thus, many communication services are emerging to supply this rapid increase of demand. Many of the services that are provided are very similar in nature. That is where the Rapid Realization of Communication Services System comes into play. First of all, it will use a model-driven approach to software development, allowing engineers to implement a series of models in a matter of minutes, worrying about coding at a minimum. It will provide an easy to use graphical interface to create these models and all the relationships among them. Finally, they can be then translated and executed, allowing great flexibility and functionality to the developer.

1.2 Scope of System

The system will target developers of communication models, as well as end users executing the models and the end users interacting with the series of calls made by the first user. The system requires the existence of a Communications Virtual Machine. The program then sits over this machine and performs communication related calls to be executed by it. For the sake of this project Skype will be used as a replacement for the CVM.

1.3 Limitations of Current Systems

Currently there are many services that provide communication capabilities to users. However, many of them do not offer automated sequence of services. If a user needs to send a chat message to a couple of people, to remind them a meeting is going to take place, send them a file containing the meeting material, and invite them to the video conference, he has to do a

lot of work. If a developer attends to his needs, he can implement this functionality. However, if this user now needs something else, it becomes very inefficient to constantly change his needs. RRCommSSys provides a graphical modeling environment where the developer can implement any kind of communication schema. The system will, in this way decrease development speed, with the ability to save and load previously stored models, increase consistency within models, and are exported in a very portable format. It is a very elegant solution for flexible communication needs.

1.4 Analysis and Design Methodology

The Unified Software Development Process (USDP) was used in the development of this project. 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 helped refine the final product as we got to know much better the specific implementation platforms, namely Eclipe GMF and 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: MVC and Pipe and Filter, whereas the design patterns: Abstract Factory, command, Façade.

We used the UML 2 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.

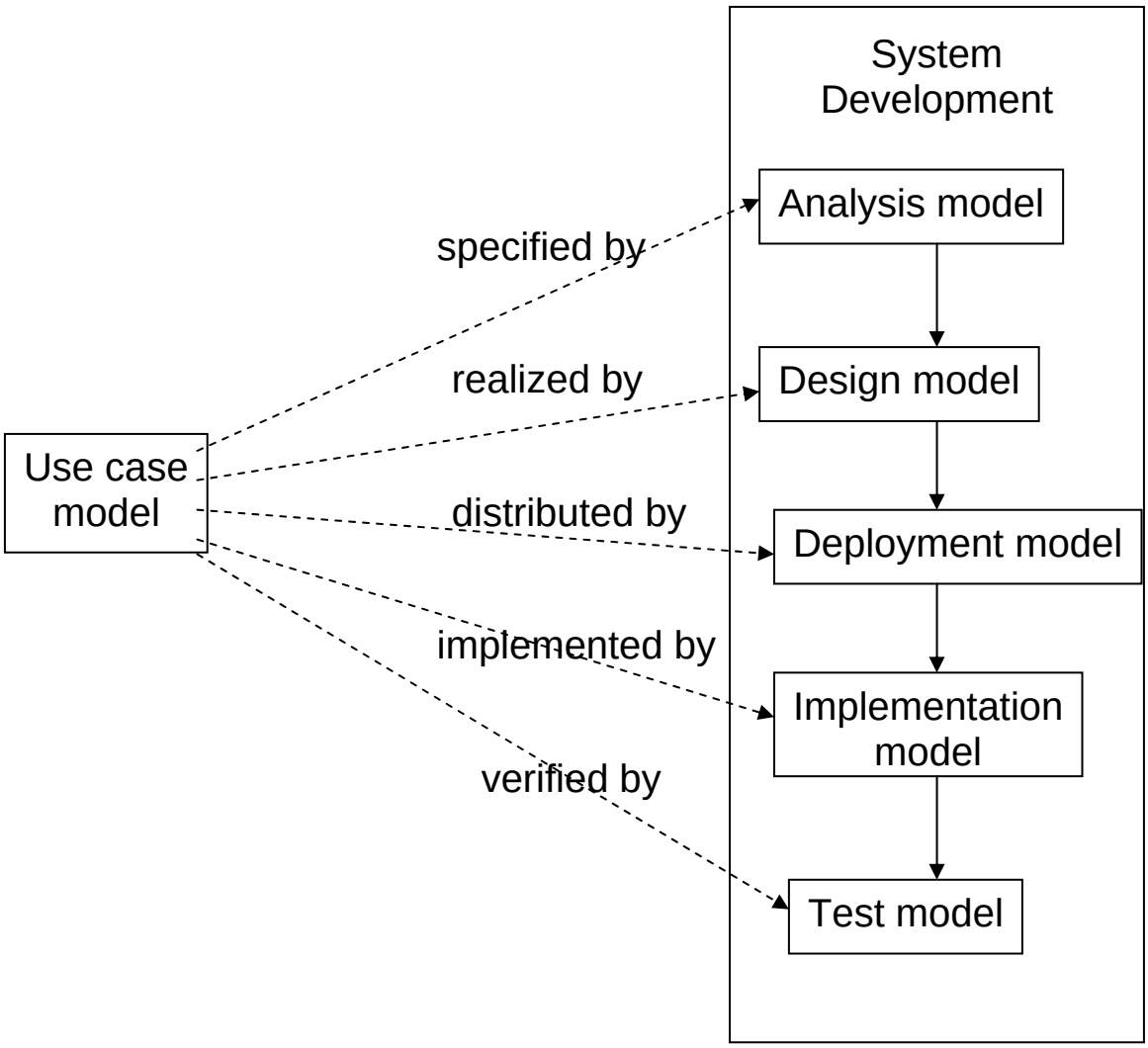


Fig 1.1 Unified Software Development Process [2]

The USDP approach was complemented with the use of the DMSD approach. The main reason for using the Model Driven Software Development (MDSD) was to increase the development speed. Through automation we are able to generate large amount of run-able code from formal models. This approach also allows for a higher quality of code, as some of the code generation is automated it the human error factor I removed from the equation and errors are less likely to appear later on in the future. The MDSD approach also aids in the management of complexity, it allows for 'programming' r configuration on a more abstract level. Reusability was another big reason was choosing this approach, once all our architectures, transformations, and modeling languages have been defined the 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.

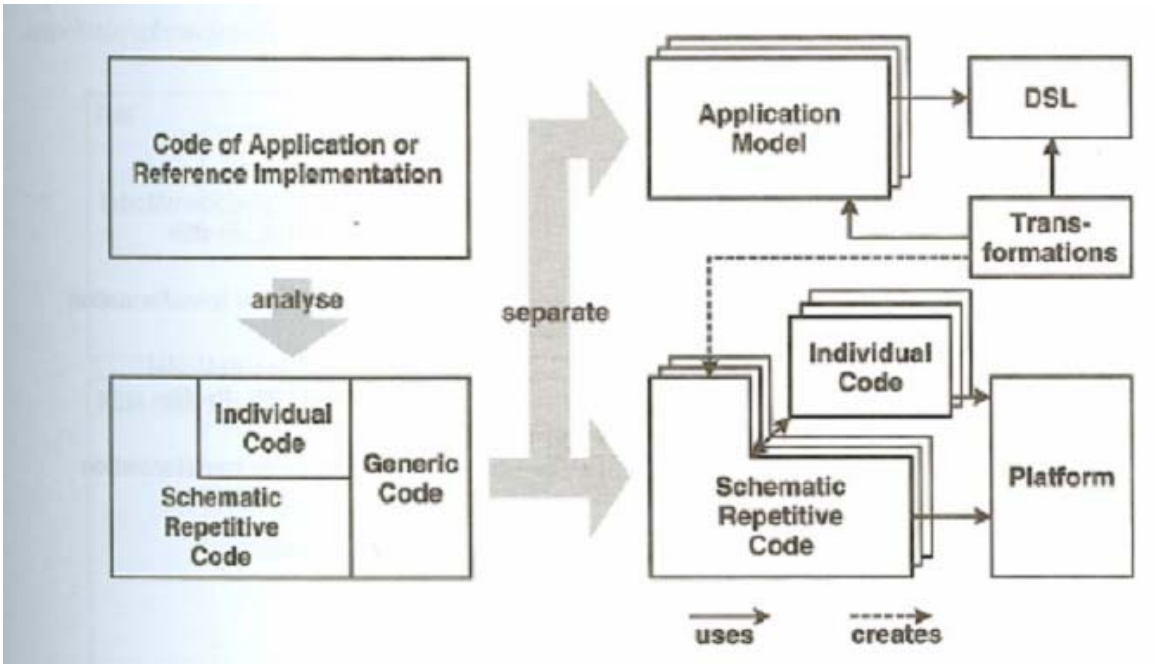


Fig. 1.2 Model Driven Software Development

The process is combination of (2) : (1) development processes basically use case driven as each model defines a certain aspect of the use cases as the development process goes on. (2) it uses MDSD to guarantee an increased development speed, better software quality, and reusability.

1.5 Definitions, Acronyms, and Abbreviations

RRCommSSys – Name of this project – Rapid Realization of communication Services System.

VE - Visual Environment**.**

GEF - Graphical Editing Framework

GMF - Graphical Modeling Framework

CVM - Communication Virtual Machine.

CML - Communication Modeling Language.

1.6 Overview of Document

The rest of this document consists of more detailed information about the development process of RRCommSSys. This is broken down into seven more chapters:

- **Chapter 2** describes the project plan for the development of RRCommSSys. The team roles, hardware/software requirements, and the work break-down are all described.
- **Chapter 3** has the functional and non-functional requirements of the system, and the system models. The latter include Scenarios and use cases, Object and Dynamic models, and the user interface.
- **Chapter 4** discusses the proposed system architecture in terms of models and metamodels. This chapter also covers the subsystem decomposition of RRCommSSys application.
- **Chapter 5** presents the object design in terms of static and dynamic models. It overviews the objects interactions as well as detailed class diagrams.
- **Chapter 6** discusses the implementation and describes the platform specific models used to run the application.
- **Chapter 7** includes a glossary of terms used in this document for the general reader.
- **Chapter 8** contains the appendixes consisting of supporting documentation and visual aids for the previous chapters.
- **Appendix A** – contains the use cases diagrams of the project.
- **Appendix B** – Use Cases with Nonfunctional Requirements
- **Appendix C** – Class diagrams for the analysis model.
- **Appendix E** – Detailed class diagrams showing attributes and methods for each class.
- **Appendix F** – Class interfaces, attributes and methods of the classes implemented in this project.
- **Appendix G -** Diary of Meetings and Tasks.

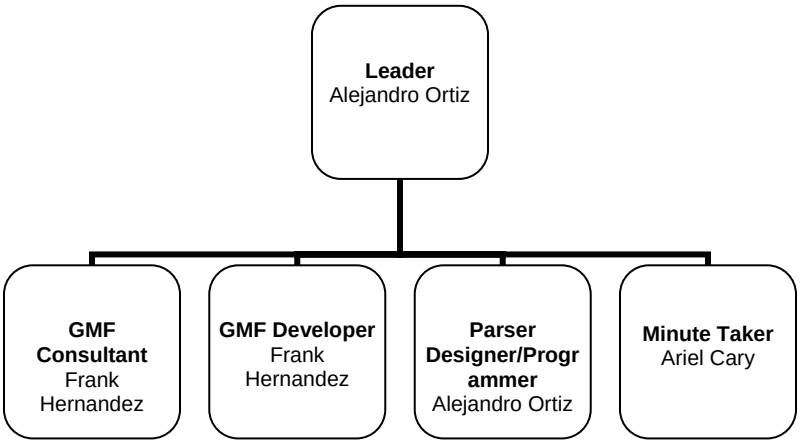
2. Project Plan

This section discusses the fundamental structure of the Rapid Realization of Communication Services System (RRComSSys for short) development plan. This includes the different roles involved in creating this project, the hardware and software requirements important to its functionality and maintenance, and the milestones and deliverables produced during each one of the phases of the project.

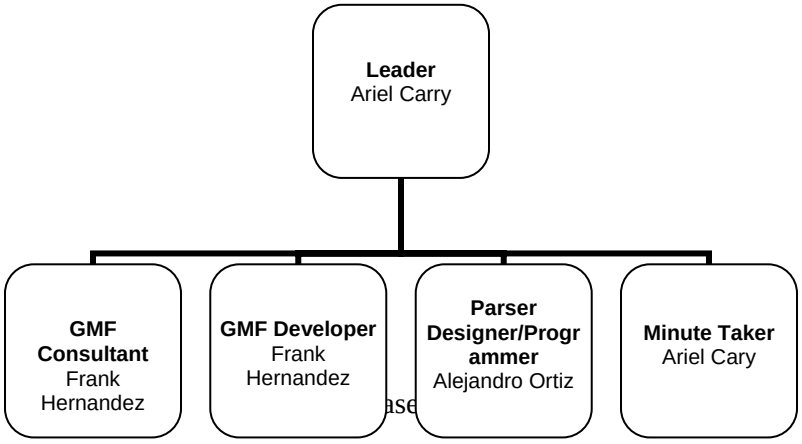
2.1 Project Organization

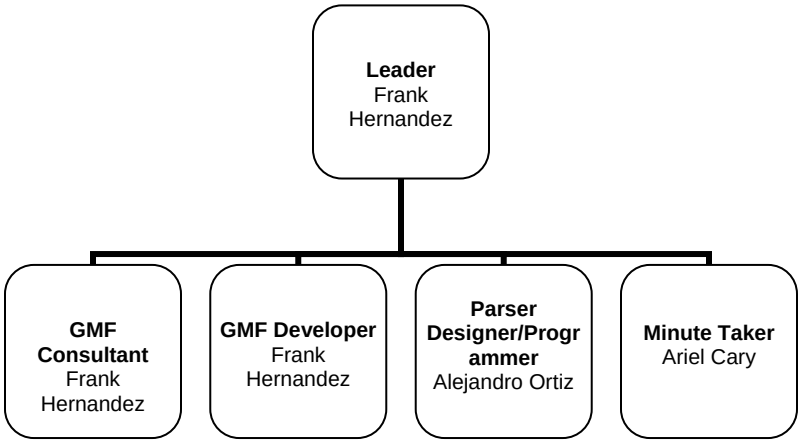
The following figure displays the hierarchy of the roles involved in the RRComSSys project throughout the first phase of development:

Phase I:



Phase II





Leader – Oversees all project tasks and ensures all milestones are reached. He/she manages group meetings, answers questions regarding the project, and assigns work to each member of the group.

GMF Developer – Works developing a product that is dependable, usable, maintainable, and efficient. His tasks are based on the designing of the Visual Environment (VE) class structure. The GMF developer will design the structure to be use by the Eclipse modeling environment that generates the VE.

GMF Consultant – Handles any question specific to the Eclipse GMF. Go-to guy when a problem arises that one of the developers can't easily solve.

Parse Designer/Programmer – Handles the design of the structure of the X-CML parser. Will also handle the main concerns with the implementation of the parser.

Minute Taker – Keeps a documented journal of all meetings. These journals include time and date, attendance, and the topics discussed during each meeting. He distributes this information to the rest of the group by the end of the week.

2.2 Hardware/Software Requirements

Hardware needed

1. Processor: 1Ghz or faster
2. Memory: 512 MB of RAM
3. Hard Drive: 40 GB

Software needed

1. Windows XP Professional
2. Microsoft Word
3. Rational Rose
4. Microsoft Project
5. Microsoft PowerPoint
6. Eclipse 3.2
7. Graphical Editing Framework (GEF)
8. Graphical Modeling Framework (GMF)
9. Eclipse Modeling Framework (EMF)
10. Skype
11. Xerces-J-bin.1.4.4
12. Xerces-J-bin.2.9.0

2.3 Work Breakdown

Tasks and Milestones

The tasks in the development of RRCommSSys were divided into three milestones: Milestone 1 consisted of the completion of the Use Case Phase and the Analysis Phase. This resulted in a software requirements document handed in to the client. It also covered Object and Dynamic models. Milestone 2 consisted of the completed of Design Phase and started on the model driven software development approach. After generating models and code, the necessary transformations will be made. This resulted in a design document handed in to the client. Finally, Milestone 3 consisted of the completion of the Testing Phase as well as the completion of the entire project. Below are the list of tasks, please refer to Appendix A for a more graphical representation. Refer to Appendix H for project schedule and Appendix I for diary.

3. Requirement Elicitation and Analysis

3.1 Overview

The system essentially allows the development of communication schemas using a declarative Communication Modeling Language (CML) as proposed in [1], in a user-friendly environment, aiming at rapid development and deployment. The CML language is defined by a grammar that describes the allowed configurations of the communication schemas. A communication scenario, that is a concrete instance of a communication schema, is executed by a Communication Virtual Machine (CVM). The communication schemas are developed by one type of user, which we call CVM developer.

Later, a CVM end-user runs the communication schema for which the system creates an instance with the information provided in the abstract model. For this purpose, the system firsts validates the model to check that it complies with the CML grammar. Second, if some information is missing for the instantiation, such as the participants, the capabilities of the communication device each person is attached to, the type of data to be transferred, etc., the system requests the CVM end-user to provide such information to execute the model.

Once the communication schema is validated and instantiated, the system executed the appropriate calls to realize the communication over a target communication platform.

3.2 Functional Requirements

The system shall:

1. Allow developers to builds communication models based on the CML language in a graphical environment.
   Use cases: All use cases related to Create Terminal and Non-terminal shapes.
   Use Case ID(s): 1.8_CrtTerm , 1.7_CrtFormNTerm, 1.6_CrtDvcNTerm, 1.5_CrtMdaAtchNTerm, 1.4_CrtRmtNTerm, 1.3_CrtLclNTerm, 1.2_CrtCnctnNTerm, 1.1_CrtUsrNTerm.

2. Provide functionality for dragging and dropping shape representations of CML Terminals onto a canvas for composing CML Non-terminals.

Use cases: Create Terminal.

Use Case ID(s): 1.8_CrtTerm.

Constraint: The system must present a user-friendly graphical interface with a drag and drop interface for declaring terminals and non-terminals. Easy to use less than 30 seconds to learn.

3. Provide functionalities for connection the various CML Terminals among one another where connections are allowed.

   Use cases: All use cases related to Create Non-terminals.

   Use Case ID(s): 1.7_CrtFormNTerm, 1.6_CrtDvcNTerm, 1.5_CrtMdaAtchNTerm, 1.4_CrtRmtNTerm, 1.3_CrtLclNTerm, 1.2_CrtCnctnNTerm, 1.1_CrtUsrNTerm.

   Constraints: The system must implement the constraints of the CML language so that the CVM developer is allowed to build only valid CVM communication models.

4. Allow developers to fill in the properties of Terminal shapes, such as person name, contact ID, device capabilities, media type, file location, and so forth.

   Use cases: Create Terminals.

   Use Case ID(s): 1.8_CrtTerm.

   Constraints: The system must clearly identify the mandatory items to be filled in when a communication instance fails to get executed due to missing data.

5. Allow developers to save communication models to file system. Likewise, the system shall be able to load saved models and display them on the canvas.

   Use cases: Save Model, Load Model.

   Use Case ID(s): 1.9_SaveMdl, 1.10_LoadMdl.

   Constraints: The system must always save/load the model to/from disk unless external conditions prevent doing so, e.g. disk full, disk failure.

6. Execute the communication model by instantiating the communication schema defined by the CVM developer.

   Use cases: Transform Model, Execute Model.

   Use Case ID(s): 2.1_SchTransf, 2.2_ExecuteMdl.

   Constraints: The validation of the communication schema must be completed in no more than 5 seconds for models having on average 100 shapes or less.

7. Request any missing data in the communication schema prior to its execution.

   Use cases: Execute Model.

   Use Case ID(s): 2.2_ExecuteMdl.

   Constraints: The system must clearly identify the mandatory items to be filled in when a communication instance fails to get executed due to missing data.

8. The types of communications supported by the system are:
   a. Voice call
   b. Instant Messaging

   Use cases: Create Voice Call Communication Model, Create Chat Message Model.

   Use Case ID(s): 1.11_CrtVoiceClCommMdl, 1.12_CrtChatCommMdl.

   Constraints: The communication model is executed by making calls to the Skype Internet-based telephony platform.

9. Security Use Case: Communication model consistency.

   Use cases: Communication Model Consistency.

   Use Case ID(s): 1.13_CommMdlConstcy.

   Constraints: The system must validate if a person is bound more than once in a communication model before executing it.

3.3 Non functional requirements

The user level requirements not directly related to functionality are:

1. Usability
   a. The system must present a user-friendly graphical interface for the development of communication schemas. The system shall be friend enough that it takes no more than 30 seconds to figure out.
2. Reliability
   a. The system must implement the constraints of the CML language so that the CVM developer is allowed to build only valid CVM communication models.
3. Performance
   a. The validation of the communication schema must be completed in no more than 5 seconds for models having on average 100 shapes or less.
4. Supportability

a. The system must clearly identify the mandatory items to be filled in when a communication instance fails to get executed due to missing data.

5. Implementation
    a. The software application must be developed using the Model-Driven Software Development approach. In particular, the Eclipse GMF component.
    b. The communication model is executed by making calls to the Skype Internet-based telephony platform.

3.4 System Models

Use case model, Dynamic model and object model all consist for the System models. Finally User interface mockups are shown in Appendixes A-G .

3.4.1 Use Case Model

Use Case Model

Description: This use case diagram shows all of the use case that will be implemented in this project. It shows the possible actions that each kind of user can attempt in this system. It also shows both of the main subsystems. Refer to figure A.1 in Appendix A.

3.4.2 Object Model

Minimal Class Diagram Of The Modeling Construction

Description The classes in this diagram represent all the non-terminal and terminal symbols of the CML grammar, which is used in the creation of communication models in RRCommSSys. This is a version compliant to the UML profile for the purpose of a better understanding of GMF during the design phase. Refer to figure C.1 in Appendix C and figure F.1 Appendix F.

Class Diagram Communication Modeling Environment

Description: The CME subsystem contains all the classes that are required for the modeling environment. This diagram show all the classes as well as all the all the dependencies. It also display the Model-View-Controller architecture pattern. This pattern is inherited from Eclipse. Refer to figure C.3 in Appendix C and figure F.7 Appendix F.

Class Diagram For The Schema Transformation Engine

Description: This is the class diagram for the Schema transformation Engine. Here are all the classes that will be implemented for the transformation of schemas into instances. And architecture pattern of Pipe and Filter is used for controlling of the stream of data between parsers. Also a Singleton design pattern is used to limit the instances of our controller to one. Refer to figure C.4 in Appendix C and figure F.4 Appendix F.

Class Diagram Synthesis Engine Subsystem.

Description: This is the class diagram for the Synthesis Engine subsystem. This are the classes that will be implemented for the of the execution of the XCML. The Command design pattern is used for more control over the calls that we will be making. Also the Abstract Factory design pattern is used to allow in future development the replacement of the Skype API for another platform, as well as to allow the system to run on multiple operating systems. Refer to figure C.5 in Appendix C and figure F.5 Appendix F.

Repository Package Class Diagram

Description: This is the class diagram for the repository that will be implemented on our project. This repository will hold the metadata as well as the information for parsing our files. Refer to figure C.5 in Appendix C and figure F.6 Appendix F

3.4.3 Dynamic Model

Sequence For a Schema Transformation.

Description: This sequence diagram shows the sequence of object involved in the execution of a schema. It shows each of the object that are affected in this execution .Refer to figure D.3 in Appendix D.

Sequence For an Instance Transformation.

Description: This sequence diagram shows the sequence of object involved in the execution of an instance. It shows each of the object that are affected in this execution. Refer to figure D.4 in Appendix D.

Sequence For Voice Call.

Description: This sequence diagram shows the sequence of object involved in the creation of a voice call model. It shows each of the object that are affected in the creation of this model. Refer to figure D.5 in Appendix D.

Sequence For Chat Message.
Description: This sequence diagram shows the sequence of object involved in the creation of a chat message model. It shows each of the object that are affected in the creation of this model. Refer to figure D.6 in Appendix D.

Sequence For Create Local Non-Terminal.
Description: This sequence diagram shows the sequence of object involved in the creation of a non-terminal in the modeling environment. It shows each of the object that are affected in the creation of this model. Refer to figure D.7 in Appendix D.

Sequence For Create Connection.
Description: This sequence diagram shows the sequence of object involved in the creation of a connection in the modeling environment. It shows each of the object that are affected in the creation of this model. Refer to figure D.8 in Appendix D.

Sequence For Create Terminal.
Description: This sequence diagram shows the sequence of object involved in the creation of a terminal in the modeling environment. It shows each of the object that are affected in the creation of this model. Refer to figure D.9 in Appendix D.

Sequence For Save File GMF.
Description: This sequence diagram shows the sequence of object involved in the saving of a model file. It shows each of the object that are affected in the saving of a model. Refer to figure D.10 in Appendix D.

3.4.4 User Interfaces

Load Request Form.
Description: This is the form that will first appear when the application is run. If the user clicks the 'Browse' button it will bring about the file open dialog. Refer to Figure E.1.1 in Appendix E.

Impute Request Form.
Description: This is the form that will be displayed if the schema that is executed is missing some information, that is if is not an instance. Refer to figure E.1.2 in Appendix E.

File System open file Dialog.

Description: This is the window that will prompt the user for the file to load. This is handled by the operating system. Refer to figure E.1.3 in Appendix E.

Visual Modeling Environment
Descrition: This is what the modeling environment looks like. This is where the communication models will be created. It is a matter of drag and drop the shape into the canvas and connect them. Refer to figure E.1.4 in Appendix E.

3.5 Validation of the Analysis Model

3.5.1 Test Cases

Identifier:	TC-1.1_CrtUsrNTerm
Owner/Creator:	Frank Hernandez
Version:	V2
Name:	Test Create User Schema Non Terminal Use Case.
Requirement ID:	1.1_CrtUsrNTerm
Purpose:	The purpose of this test case is to test the Create User Schema Non Terminal Use Case
Dependencies:	None
Environment/ Configuration:	None
Initialization:	The Communication Modeling Environment must be initialized. Terminals and non-terminals for a Local non-terminal and Connection non-terminal must be on the canvas. Local non-terminal is composed of one Connection terminal connected to a Medium terminal connected to a Device connected isAttached connected to a local Person.
Finalization:	If the modeling fails the application must be reloaded.
Actions:	Try to connect the Device terminal to the connection non-terminal.
Input data:	Attempts to perform the actions specified on the ‘Action’ section.
Expected results:	The Connection is drawn onto the canvas.

Identifier:	TC-1.2_CrtCnctnNTerm

Owner/Creator:	Frank Hernandez
Version:	V2
Name:	Test Create Connection Non-Terminal
Requirement ID:	1.2_CrtCnctnNTerm
Purpose:	The purpose of this test case is to test the Create Connection Non-Terminal Usecase.
Dependencies:	None
Environment/ Configuration:	None
Initialization:	The Communication Modeling Environment must be initialized prior the execution of this test. Non-terminal for a MediaAttached non-terminal must be on the canvas. One or more Remote non-terminal must be on the canvas. A MediaAttached is composed of at least one Medium terminal. A Remote non-terminal consist of Person connected to isAttached to Device.
Finalization:	If no shape is drawn the CME must be checked before any other test takes place.
Actions:	Drag Connection terminal onto the canvas. Drag connection, connecting Medium to the Connection terminal. Drag MedToCon and connect Conenction to a Medium
Input data:	Attempts to perform the actions specified on the 'Action' section.
Expected results:	The Connection terminal is drawn on the canvas. The connections are drawn between the Connection terminal and the MediaAttached terminal.

Identifier:	TC-1.3_CrtLclNTerm
Owner/Creator:	Alejandro Ortiz
Version:	V1
Name:	Test Create Local Non-Terminal
Requirement ID:	1.3_CrtLclNTerm
Purpose:	The purpose of this test case is to test the Create Local Non-Terminal Use Case.
Dependencies:	None
Environment/ Configuration:	None
Initialization:	The Communication Modeling Environment must be initialized prior the

	execution of this test.
Finalization:	If no shape is drawn the CME must be checked before any other test takes place.
Actions:	Drag Person terminal onto the canvas. Drag isAttahced terminal onto the canvas. Drag PtoIACon connecting the Person terminal to the isAttached terminal. Drag a Device terminal onto the canvas. Drag IAToDevice connection and connect the Device to the isAttached.
Input data:	Attempts to perform the actions specified on the 'Action' section.
Expected results:	The Person terminal is drawn on the canvas. The isAttached terminal is drawn on the canvas. The connections are drawn between the Person terminal and the isAttached terminal.

Identifier:	TC-1.4_CrtLclNTerm
Owner/Creator:	Alejandro Ortiz
Version:	V1
Name:	Test Create Remote Non-Terminal
Requirement ID:	1.4_CrtRmtNTerm
Purpose:	The purpose of this test case is to test the Create Remote Non-Terminal Use Case.
Dependencies:	None
Environment/ Configuration:	None
Initialization:	The Communication Modeling Environment must be initialized prior the execution of this test.
Finalization:	If no shape is drawn the CME must be checked before any other test takes place.
Actions:	Drag Person terminal onto the canvas. Drag isAttahced terminal onto the canvas. Drag PtoIACon connecting the Person terminal to the isAttached terminal.
Input data:	Attempts to perform the actions specified on the 'Action' section.
Expected results:	The Person terminal is drawn on the canvas. The isAttached terminal is drawn on the canvas. The connections are drawn between the Person terminal and the isAttached terminal.

Identifier:	TC-1.5_CrtMdaAtchNTerm
Owner/Creator:	Frank Hernandez
Version:	V1
Name:	Test Create MediaAttached Non-Terminal
Requirement ID:	1.5_CrtMdaAtchNTerm
Purpose:	The purpose of this test case is to test the Create MediaAttached Non-Terminal Use Case.
Dependencies:	None
Environment/ Configuration:	None
Initialization:	The Communication Modeling Environment must be initialized prior the execution of this test. Non-terminal for one or more Form non-terminal must be on the canvas already. A Form non-terminal consists of one or more Medium terminals.
Finalization:	If no shape is drawn, the CME must be checked before any other test takes place.
Actions:	Drag Medium terminal onto the canvas. Drag connection line connecting the Medium terminal to the Connection terminal.
Input data:	Attempts to perform the actions specified on the 'Action' section.
Expected results:	The Medium shape is drawn on the canvas, as well as the connection to the Form non-terminal.

Identifier:	TC-1.6_CrtDvcNTerm
Owner/Creator:	Ariel Carry
Version:	V1
Name:	Test Create Device Non-Terminal Use Case.
Requirement ID:	1.6_CrtDvcNTerm
Purpose:	The purpose of this test case is to test the Create Device Non-Terminal
Dependencies:	None

Environment/ Configuration:	None
Initialization:	The Communication Modeling Environment must be initialized prior the execution of this test.
Finalization:	If no shape is drawn, the CME must be checked before any other test takes place.
Actions:	Drag Capability terminal onto the canvas. Drag Device terminal onto the canvas. Drag CapToDev line connecting the Capability terminal to the Device terminal.
Input data:	Attempts to perform the actions specified on the 'Action' section.
Expected results:	The Capability shape and the Device shape are drawn on the canvas, as well as the connection from Device to Capability.

Identifier:	TC-1.8_CrtTerm
Owner/Creator:	Frank Hernandez
Version:	V1
Name:	Test Create Terminal
Requirement ID:	1.8_CrtTerm
Purpose:	The purpose of this test case is to test the Create Terminal Use Case.
Dependencies:	None
Environment/ Configuration:	None
Initialization:	The Communication Modeling Environment must be initialized prior the execution of this test.
Finalization:	If no shape is drawn, the CME must be checked before any other test takes place.
Actions:	Select Terminal from the shape palette in the CME. Drag the Terminal onto the canvas.
Input data:	Attempts to perform the actions specified on the 'Action' section.
Expected results:	The Terminal shape is drawn on the canvas.

Identifier:	TC-1.9_SaveMdl
Owner/Creator:	Frank Hernandez
Version:	V1
Name:	Test Save Model Use Case.
Requirement ID:	1.9_SaveMdl
Purpose:	The purpose of this test case is to test the Save Model Use Case
Dependencies:	None
Environment/ Configuration:	None
Initialization:	The Communication Modeling Environment must be initialized prior the execution of this test.
Finalization:	If the save fails the application must be checked.
Actions:	Select Save Option from the File Menu. Enter Filename.
Input data:	Attempts to perform the actions specified on the 'Action' section.
Expected results:	The file is saved.

Identifier:	TC-1.10_LoadMdl
Owner/Creator:	Alejandro Ortiz
Version:	V1
Name:	Test Load Model Use Case.
Requirement ID:	1.10_LoadMdl
Purpose:	The purpose of this test case is to test the Load Model Use Case
Dependencies:	None
Environment/ Configuration:	None
Initialization:	The Communication Modeling Environment must be initialized prior the execution of this test.
Finalization:	If the load fails the application must be checked.

Actions:	Select Load Option from the File Menu. Select the file and load it.
Input data:	Attempts to perform the actions specified on the ‘Action’ section.
Expected results:	The file is loaded.

Identifier:	TC-2.1_SchTransf
Owner/Creator:	Ariel Carry
Version:	V1
Name:	Test Schema Transformation Use Case.
Requirement ID:	2.1_SchTransf
Purpose:	The purpose of this test case is to test the Schema Transformation Use Case
Dependencies:	None
Environment/ Configuration:	None
Initialization:	The Schema Transformation Environment most be initialized
Finalization:	If the transformation fails the application must be reloaded.
Actions:	Convert an incomplete instance into a complete one.
Input data:	Attempts to perform the actions specified on the ‘Action’ section.
Expected results:	The schema is transformed into am instance.

Identifier:	TC-2.2_ExecuteModel
Owner/Creator:	Ariel Carry
Version:	V1
Name:	Test Execute Model Use Case.
Requirement ID:	2.2_ExecuteModel
Purpose:	The purpose of this test case is to test the Execute Model Use Case
Dependencies:	None

Environment/ Configuration:	None
Initialization:	The Synthesis Engine must be initialized
Finalization:	If the execution fails the application must be reloaded.
Actions:	Execute instance to Skype calls.
Input data:	Attempts to perform the actions specified on the ‘Action’ section.
Expected results:	The instance is executed and Skype calls are made.

Identifier:	TC-1.11_CrtVoiceClCommMdl
Owner/Creator:	Frank Hernandez
Version:	V1
Name:	Test Create Voice Call Communication Model Use Case.
Requirement ID:	1.11_CrtVoiceClCommMdl
Purpose:	The purpose of this test case is to test the Create Voice Call Communication Model Use Case
Dependencies:	None
Environment/ Configuration:	None
Initialization:	The Communication Modeling Environment must be initialized.
Finalization:	If the modeling fails the application must be reloaded.
Actions:	1. Drag Person Terminal to the Canvas. 2. Drag isAttached Terminal to the Canvas. 3. Connect Person to isAttached. 4. Drag Device Terminal to the Canvas. 5. Connect Device to isAttahced. 6. Repeat 1-5 For next user. 7. Drag Connection Terminal to the canvas. 8. Connect Connection Terminal to both devices.
Input data:	Attempts to perform the actions specified on the ‘Action’ section.
Expected results:	All the terminals and connections are drawn on the canvas.

Identifier:	TC-1.12_CrtChatCommMdl
Owner/Creator:	Alejandro Ortiz
Version:	V1
Name:	Test Create Chat Message Model Use Case.
Requirement ID:	1.12_CrtChatCommMdl
Purpose:	The purpose of this test case is to test the Create Chat Message Model Use Case
Dependencies:	None
Environment/ Configuration:	None
Initialization:	The Communication Modeling Environment must be initialized.
Finalization:	If the modeling fails the application must be reloaded.
Actions:	1. Drag Person Terminal to the Canvas. 2. Drag isAttached Terminal to the Canvas. 3. Connect Person to isAttached. 4. Drag Device Terminal to the Canvas. 5. Connect Device to isAttahced. 6. Repeat 1-5 For next user. 7. Drag Connection Terminal to the canvas. 8. Connect Connection Terminal to both devices. 9. Drag Medium Terminal onto the canvas. 10. Connect Medium to the Connection Terminal
Input data:	Attempts to perform the actions specified on the 'Action' section.
Expected results:	All the terminals and connections are drawn on the canvas.

3.5.2 Analysis Model

Analysis Domain Model Inspection	**Yes**	**No**
Completeness: The concepts are sufficient to cover the scope of the content specified.	X	
Correctness: The description of domain concepts is accurate; the algorithms will produce the expected result, i.e. There must be both a caller and a called.	X	
Consistency: Model elements are consistent with the company's definition.	X	

Analysis Application Model Inspection	**Yes**	**No**
Completeness: The ideas expressed in each use case can be represented by the concepts and algorithms in the model.	X	
Correctness: 'Experts' agree with the attributes and behaviors assigned to each	X	

concept.		
Consistency: Where there are multiple ways to represent a concept of action those ways are equivalent.	X	

Syntax Check for Object Diagram	**Yes**	**No**
Check objects and links.	X	
Ensure that there is only one object per rectangle.	X	
Ensure that no attributes or operations are showed on minimal diagrams.	X	
Ensure that no multiplicity is shown on minimal diagrams.	X	
Ensure that note are correctly represented on the diagram.	X	

Semantic Check for Object Diagram	**Yes**	**No**
Ensure that example that there are not too many object diagrams..	X	

Semantic Check for Object Diagram	**Yes**	**No**
Ensure that example object diagrams are drawn whenever needed to clarify links.	X	
Relate the objects on the object diagram to other diagrams whose meanings the object diagrams are supposed to clarify.	X	

Syntax Checks for Sequence Diagrams	**Yes**	**No**
Check the correctness of all the objects on the sequence diagram.	X	
Check the correctness of all actors on the sequence diagram.	X	
Check object-oriented interaction.	X	
Check the message types shown in the sequence diagram.		X
Check the message of the message signatures and return values.		X
Check syntax of multiple messages.	X	
Check for multiple objects on the sequence diagram.	X	

Semantic Checks for Sequence Diagrams	**Yes**	**No**
Check the meaning behind the sequence diagram.	X	
Check the meaning behind the focus of control.	X	
Check to see if the sequence diagram depicts creation and destruction of objects.	X	
Check to see if sequence diagram is based on a pattern.		X
Check to see if there are alternative flows and create separate sequence diagrams for them.		X

Aesthetic Checks for Sequence Diagrams	**Yes**	**No**
Ensure that the sequence diagram shows a cohesive set of interactions between collaboration objects.	X	
Ensure that the sequence diagrams have sufficient notes and other annotations to explain the technicality of the diagrams.	X	
Check the number of objects.	X	
Check the number of messages.	X	

3.5.3 Structure Walkthrough

Use Case Model	Yes	No
There is a use case in the model for every use case.	X	
Every actor interacts with the specified use case.	X	
Every Use Case extends the required use case in the model.	X	
Every Use Case includes the required use case in the model.	X	
Every Use Case name in the diagram is relevant to the use cases.	X	

Dynamic Diagrams	Yes	No
There is a sequence diagram for every use case.	X	
Every noun maps to an object in the sequence	X	
Every verb maps to an interaction in the sequence diagram.	X	
Every path was followed.	X	

Object Model	Yes	No
Every class represents a noun in the use case.	X	
Connectivity is maintained from the use case to class diagram.	X	
Dependency was showed with relations.	X	

Test Case – Use Case Model	Comments	PASS	FAIL
TC-1.1_CrtUsrNTerm		X	
TC-1.2_CrtCnctnNTerm		X	
TC-1.3_CrtLclNTerm		X	
TC-1.4_CrtLclNTerm		X	
TC-1.6_CrtDvcNTerm		X	
TC-1.8_CrtTerm		X	
TC-1.9_SaveMdl		X	
TC-1.10_LoadMdl		X	
TC-2.1_SchTransf		X	
TC-2.2_ExecuteModel		X	
TC-1.11_CrtVoiceClCommMdl		X	
TC-1.12_CrtChatCommMdl		X	
TC-1.13_CommMdlConstcy		X	

Test Case – Minimal Class Diagram	Comments	PASS	FAIL
TC-1.1_CrtUsrNTerm		X	
TC-1.2_CrtCnctnNTerm		X	
TC-1.3_CrtLclNTerm		X	
TC-1.4_CrtLclNTerm		X	
TC-1.6_CrtDvcNTerm		X	
TC-1.8_CrtTerm		X	
TC-1.9_SaveMdl		X	
TC-1.10_LoadMdl		X	
TC-2.1_SchTransf		X	
TC-2.2_ExecuteModel		X	

TC-1.11_CrtVoiceClCommMdl		X	
TC-1.12_CrtChatCommMdl		X	
TC-1.13_CommMdlConstcy		X	

Test Case – Sequence Diagrams	**Comments**	**PASS**	**FAIL**
TC-1.1_CrtUsrNTerm		X	
TC-1.2_CrtCnctnNTerm		X	
TC-1.3_CrtLclNTerm		X	
TC-1.4_CrtLclNTerm		X	
TC-1.6_CrtDvcNTerm		X	
TC-1.8_CrtTerm		X	
TC-1.9_SaveMdl		X	
TC-1.10_LoadMdl		X	
TC-2.1_SchTransf		X	
TC-2.2_ExecuteModel		X	
TC-1.11_CrtVoiceClCommMdl		X	
TC-1.12_CrtChatCommMdl		X	
TC-1.13_CommMdlConstcy		X	

4. Proposed Software Architecture

The RRCommSSys 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 briefly.

4.1 Overview – Package Diagram

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

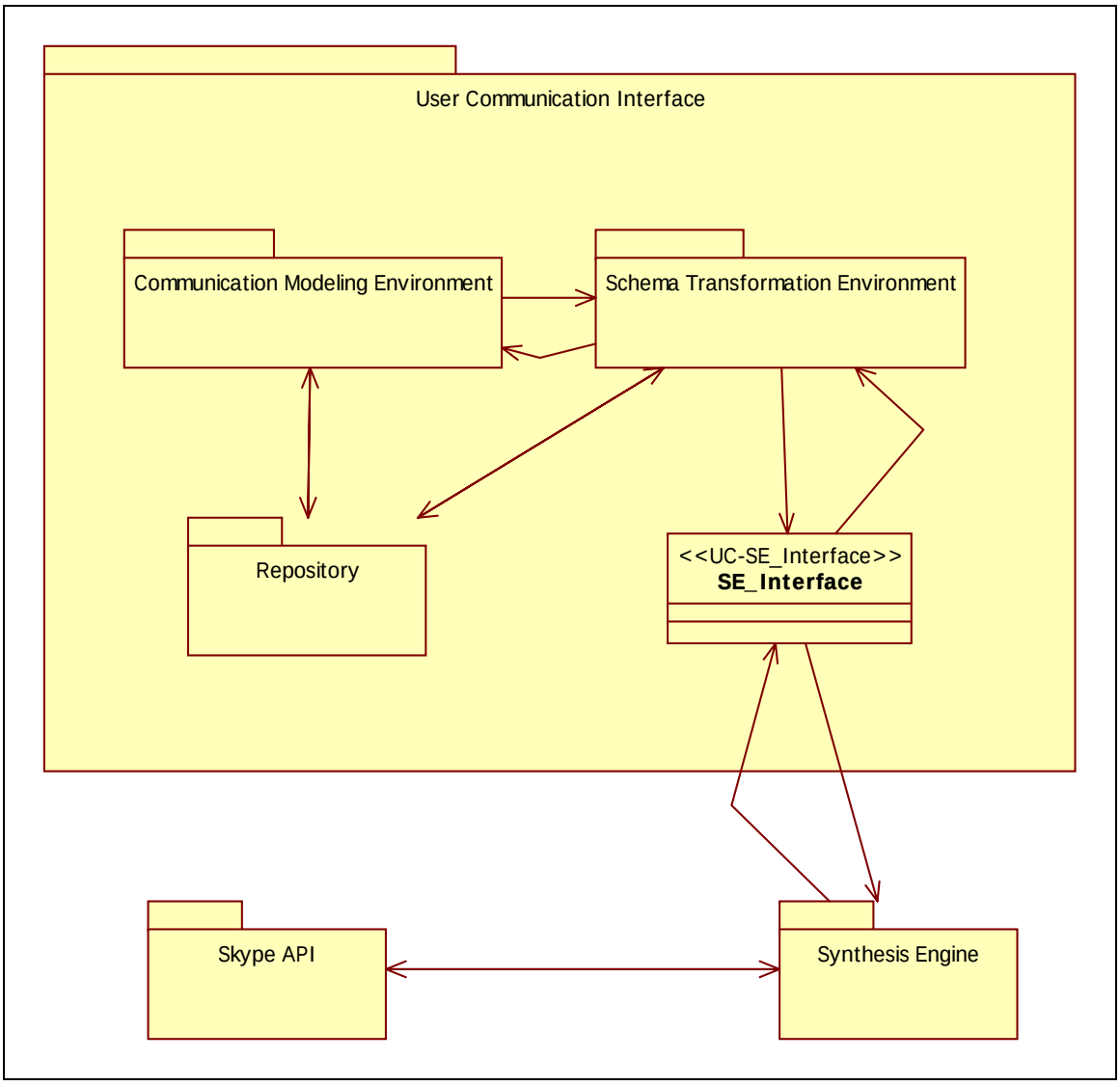


Fig 4.1 RRCommSSys Package diagram

Description: This is the package diagram for RRCommSSys. It displays the subsytem decomposition of the architecture. Also at first glance it can be seen that we will use a Repository Architecture pattern. This will hold the metadata for the models as well as the rule for our modeling languages.

Subsystems:

User Communication Interface – This is one of the main subsystems. This subsystems contains all of the subsystems required to handle the creation and the parsing of user communication models. This subsystem is composed of the Communication Modeling Environment subsystem, the Schema Transformation Environment subsystem and the Repository subsystem.

Communication 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 subsystems is inherently Model-View-Control (MVC) as it inherits this from Eclipse itself.

Schema Transformation Environment – This subsystems contains all the modules and clases related to the transformation of schemas into instances. This subsystem handles the conversion GXML into XCML that will then be interpreted for execution. It also handles the completion of any incomplete schema into a complete one, which it is then referred to as an instance.

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 Skpye.

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

RRCommSSys uses three architectural patterns justified below:

Pipe and Filter – 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 that process the models and outputs the desired format. It also simplifies 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.

Model View Controller – This pattern will handle the front-end interface with the user. It will handle user input, which will be processed by the model and later handled by the Pipe and Filter architecture. After all the processing is done, the output can be displayed as different views. In conclusion, it comprises of a Model, View, and Controller Subsystems. This pattern was chosen also because the target implementation platform GMF Eclipse is also MVC.

Façade pattern – This pattern will simplify the communication between the subsystems. It will provide a simplified interface for subsystem to communicate with one another.

4.2 Metamodel for the DSL

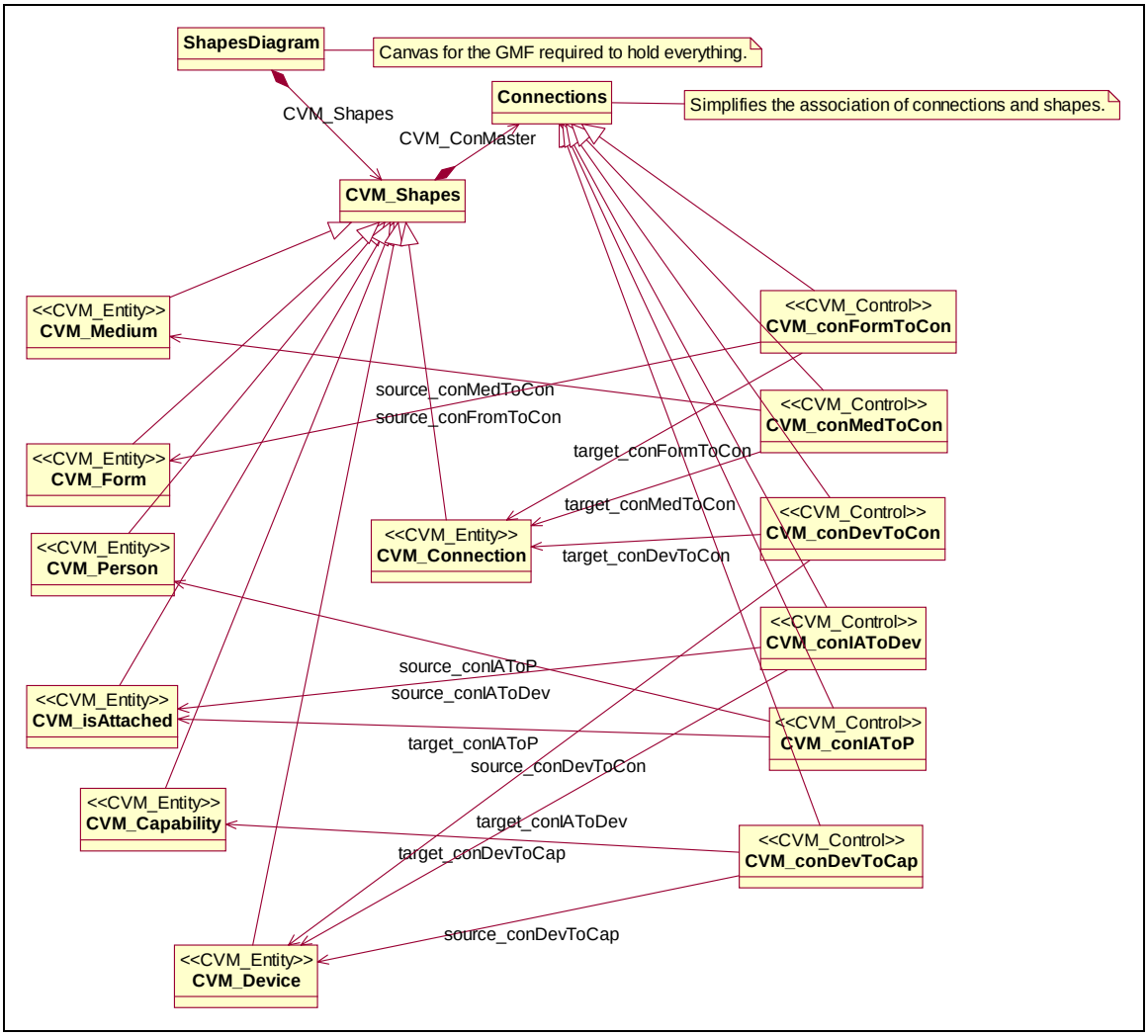


Fig 4.2. Minimal class diagram for the metamodel as input into Eclipse GMF

Description The classes in this diagram represent all the non-terminal and terminal symbols of the CML grammar, which is used in the creation of communication models in RRCommSSys.

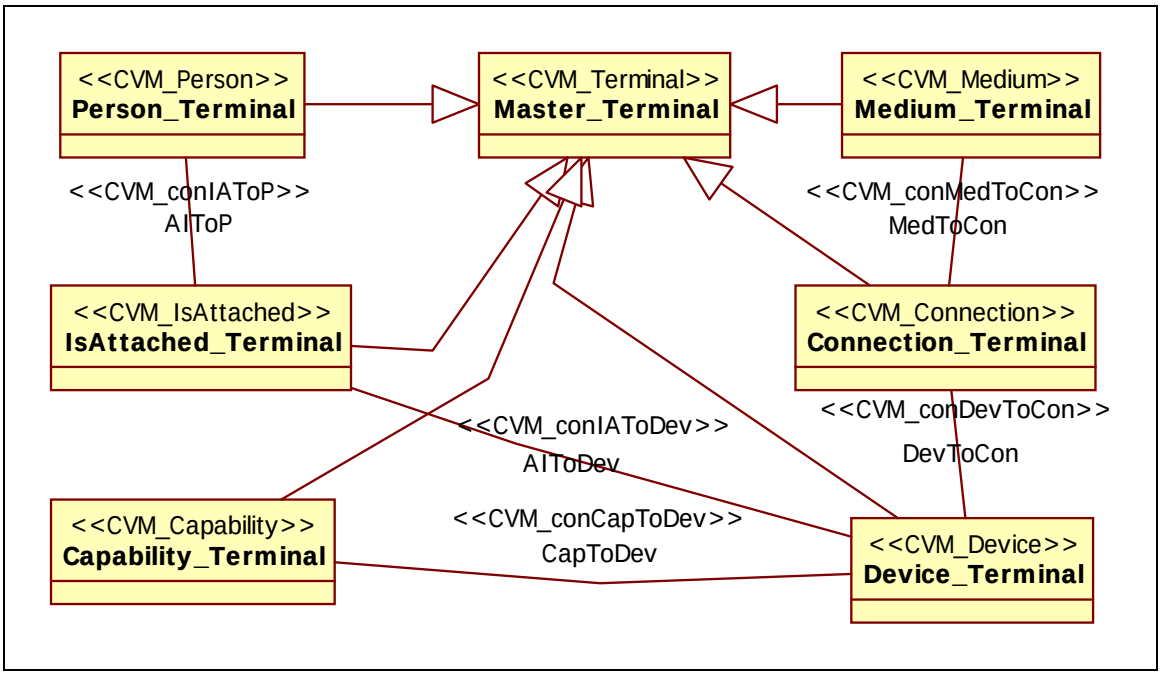


Fig 4.3. Minimal class diagram compliant the UML profile for the metamodel

Description The classes in this diagram represent all the non-terminal and terminal symbols of the CML grammar, which is used in the creation of communication models in RRCommSSys. This is a version compliant to the UML profile for the purpose of a better understanding of GMF during the design phase.

4.3 UML profiles

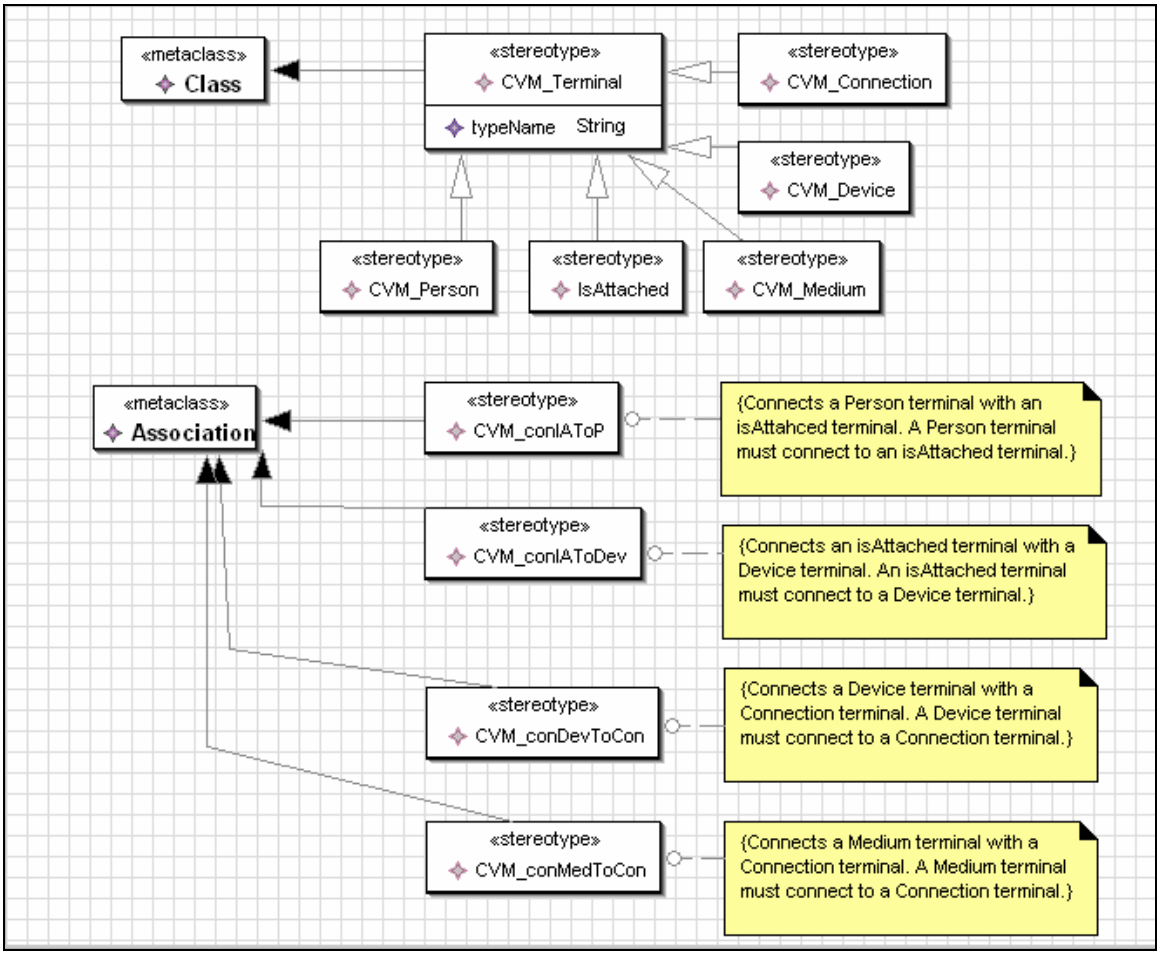


Fig 2.3 Profile For the Modeling Constructing (GMF Side)

Description: This UML 2 profile represents the metamodel. This profile gives a better representation of the metamodel. Used in the design phase to me the metamodel more manageable.

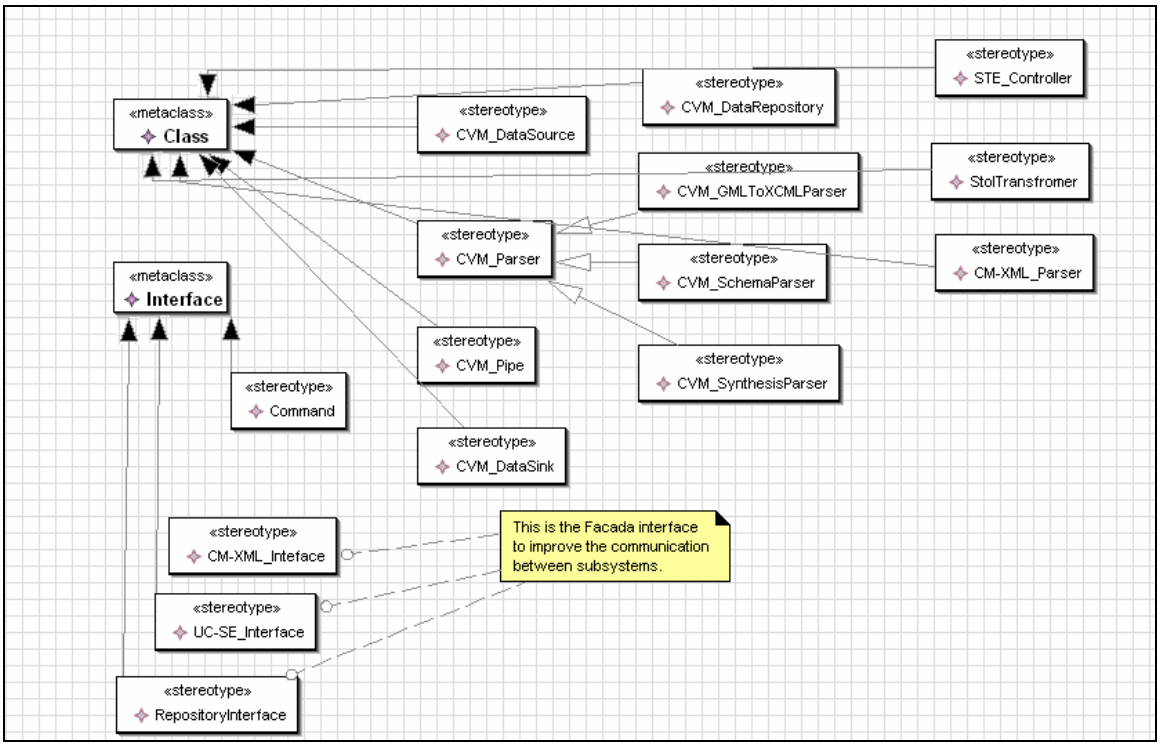


Fig. 4.4 Profile For Execution Subsystem.

Description: This profile represent the execution of a GML file. This profile is used to describe the conversion form GML to XCML and the transformation into CVM calls.

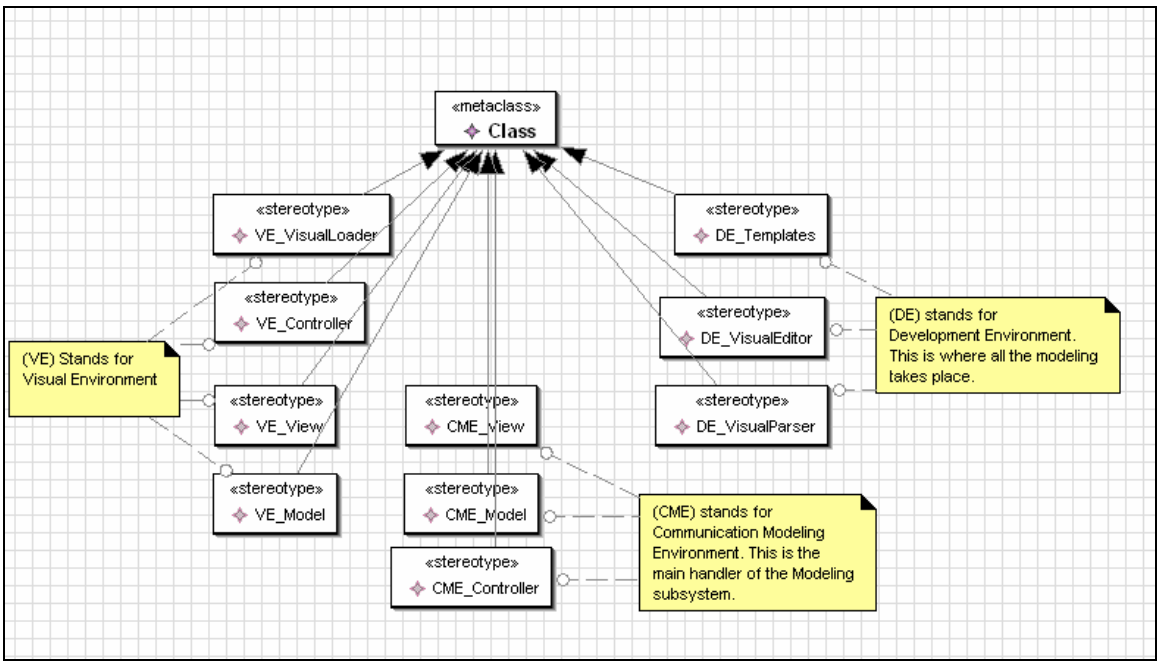


Fig 4.5 Profile For the Communication Modeling Environment.(CME)

Description: This is the profile for the Communication Modeling Environment. It describes the main prototypes of the system as well as the architecture to be used in this subsystem. The Model-View-Controller architecture is used for this subsystem.

4.4 Generative architecture

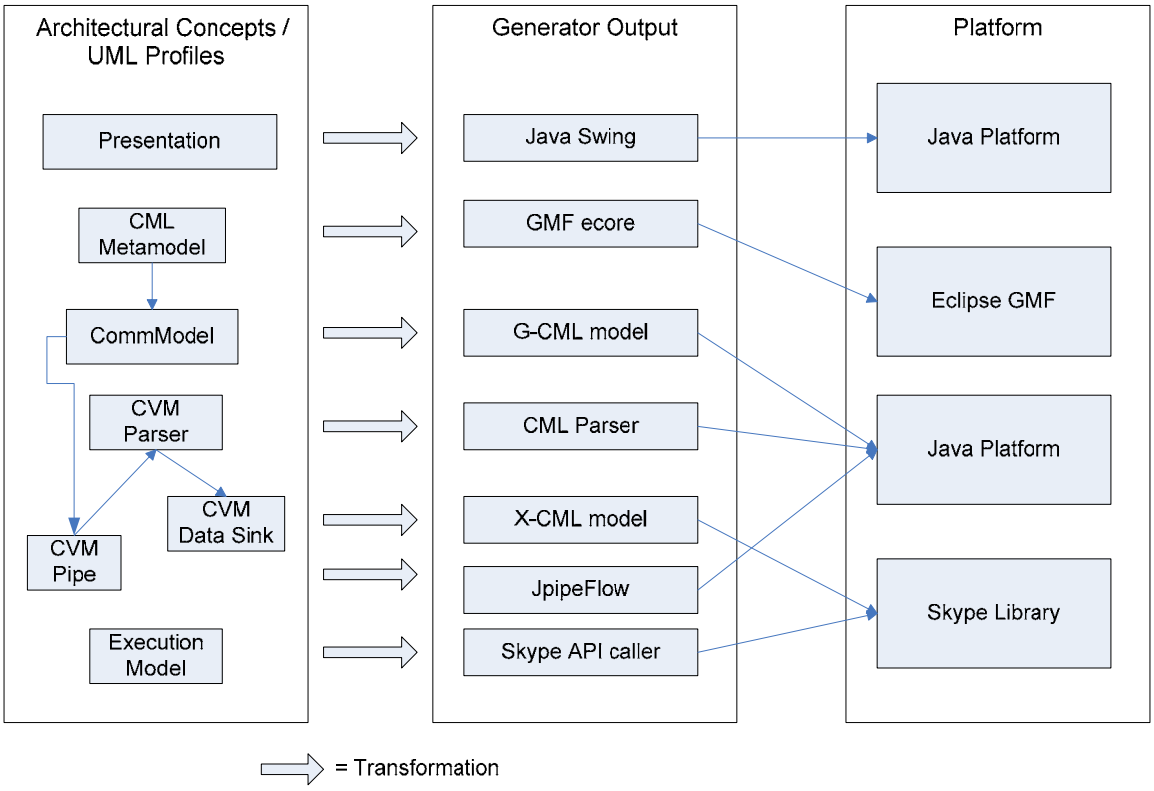


Fig. 4.5 RRCommSSys Generative Architecture

The previous figure shows the transformations carried out to the target platform. Essentially, the presentation objects are paired back to the Java Swing libraries which implement the user interface. Similarly, the model creation concepts are transformed to classes that abstract out the communication model, for example, the CML metamodel is transformed to a GMF ecore definition file, which in turn is fed to the Eclipse GMF target platform. As can also be seen on the figure, the *Data Sink* is transformed to a X-CML model, which complies with the CML language in XML format. This XML file is later used in the execution of the model by the *Skype API Caller* to make the actual call to the *Skype Library*, which renders the communication.

4.5 Subsystem Decomposition

Transformer – There will be many classes that will process data, performing a function to it and output it back. Many of these will be associated with parsing communication models from one format and turn it into another. This package is only coupled with the Pipe subsystem, which will simply feed the data to the Filter, and then receive the processed version of it back from the Filter class.

Data source – This subsystem will be the entry point to the system internally. It will start deliver the data to the Pipe subsystem, starting the chain. The data will be supplied by the Model, after it is first inputted by the user to a Controller.

Pipe – The Pipe subsystem will be in charge of delivering the data to the proper subsystems. It will receive the initial data, along with the operations needed to be performed to it, and a final destination. These operations are sequential and are composed of one or more Filters. This subsystem will ensure the operations will be made in proper order, and deliver the result to the data sink specified.

Data sink – This subsystem represents the final stage of processed input. It performs a last function to the data supplied, reflecting the original request that was made by the user.

Model – It is the core of the system. It will create the views necessary for the user to interact with the system and make a request. The Controller associated with that view will deliver the request back to the Model, which will add some extra information. This result will be delivered properly to the data source subsystem.

View – This subsystem displays the current data to the user. Each class contains a different way of displaying the same data provided by the Model, and will all initialize a controller that will capture all user interaction with the data.

Controller – There will be one Controller object for each view that is active on the system. This subsystem is in charge of receiving user input and delivering it to the Model. It is also in charge of listening for messages coming from the Model that need to be reflected on its associated view, which will be processed and delivered to the view.

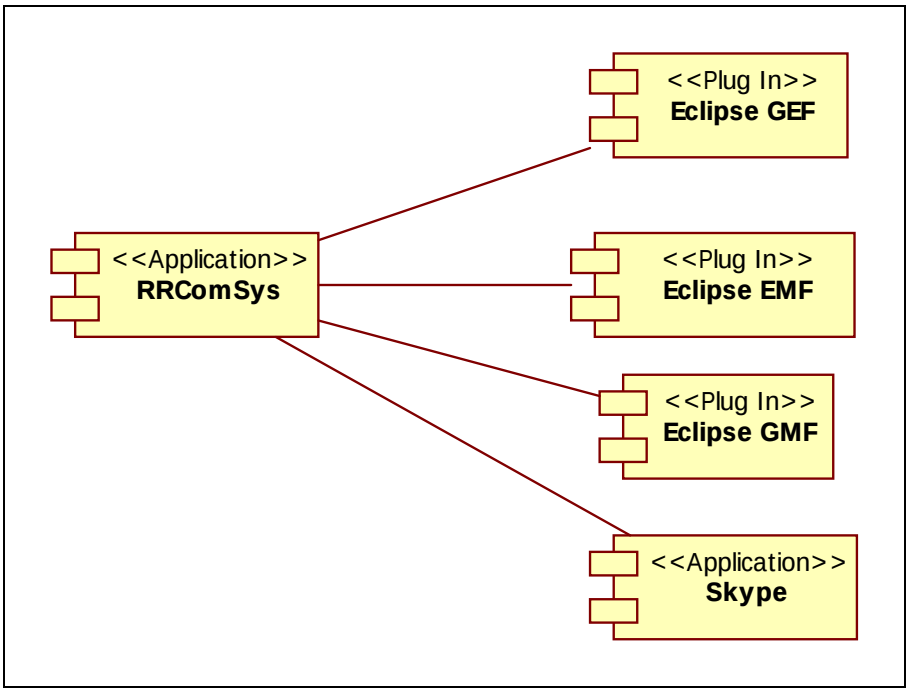


Fig 4.1 Component Diagram For RRComSys

Description: This is component diagram displaying all the PSM that are required by our application.

4.6 Validation of the System Model

4.6.1 Check List

System Checks	Yes	No
Every subsystem was touched by at least one use case.	X	
Every noun in each of the use cases can be mapped to at least one class inside one of the subsystems.	X	
Each subsystem has annotations for clarification.	X	
Check What does an operation mean? Ensure that the meaning of the operation is reflected in its name and format.	X	
All sequence diagrams touch all the subsystems.	X	
Subsystems compliant with each of the profiles.	X	
At least one of the subsystems depicts the architecture pattern used in the application.	X	
Architecture is annotated in the subsystems with notes.	X	
Check to see if the operations of a class are overloaded.	X	
Subsystem relations for a specific subsystem are displayed through the use of minimal subsystem packages.	X	
Check the name of the subsystems matches the specified name in the architecture diagram.	X	
Check for dependencies among subsystem.	X	
Check for relations between subsystems.	X	

4.6.2 Structure Walkthrough

Requirements	Yes	No
Satisfies Non-Functional requirements.	X	
Satisfies Functional Requirements.	X	

Test Case – Non Functional	Comments	PASS	FAIL
TC-1.1_CrtUsrNTerm		X	
TC-1.2_CrtCnctnNTerm		X	
TC-1.3_CrtLclNTerm		X	
TC-1.4_CrtLclNTerm		X	
TC-1.6_CrtDvcNTerm		X	
TC-1.8_CrtTerm		X	
TC-1.9_SaveMdl		X	
TC-1.10_LoadMdl		X	
TC-2.1_SchTransf		X	
TC-2.2_ExecuteModel		X	
TC-1.11_CrtVoiceClCommMdl		X	
TC-1.12_CrtChatCommMdl		X	
TC-1.13_CommMdlConstcy		X	

Test Case – Functional	Comments	PASS	FAIL
TC-1.1_CrtUsrNTerm		X	
TC-1.2_CrtCnctnNTerm		X	
TC-1.3_CrtLclNTerm		X	
TC-1.4_CrtLclNTerm		X	
TC-1.6_CrtDvcNTerm		X	
TC-1.8_CrtTerm		X	
TC-1.9_SaveMdl		X	
TC-1.10_LoadMdl		X	
TC-2.1_SchTransf		X	
TC-2.2_ExecuteModel		X	
TC-1.11_CrtVoiceClCommMdl		X	
TC-1.12_CrtChatCommMdl		X	
TC-1.13_CommMdlConstcy		X	

5. Object Design

This section will cover in detail the main view the structure of the application to be designed. It will detail the basic idea of the functionality of the software. It will also discuss some of the design patterns chosen for the implementation of the sections of RRCommSSys. This section will explain some of the reasons for choosing these patterns mainly the ones that made them a valid choice.

This section will also explain in detail the classes that will be created. It will explain their functionality and purpose. It will also show some of the behavior of the system under some user interactions.

5.1 Overview

Refer to Appendix C for minimal class diagrams.

5.1.1 Brief Class Description

Modeling Environment Classes:

EclipesMAndITemp: This class handles the operation relevant to the menus of the application. Refer to the class diagram F.7 on Appendix F.

EclipseVisualEditor: This class handles the interaction of the user and the modeling environment. Refer to the class diagram F.7 on Appendix F.

EclipseVisualParser: This class handles the conversion of visual input into GML files. Refer to the class diagram F.7 on Appendix F.

EclipseVisualLoader: This class handles the loading of the Communication Modeling Environment. Refer to the class diagram F.7 on Appendix F.

EclipseVEController: This is the controller for the visual environment. Refer to the class diagram F.7 on Appendix F.

EclipseMenuSysAndDisp: This class handles the fixed menu systems and display. Refer to the class diagram F.7 on Appendix F.

CMEController: This is the controller of the Communication Modeling Environment. Refer to the class diagram F.7 on Appendix F.

EclipseModelTransformer: The eclipse model transformer is that class that will handle the transformation from GML to XGML that is understood by Eclipse. Refer to the class diagram F.7 on Appendix F.

Model Creation Classes:

Modeling Subsystem: The modeling subsystem includes all the classes created as part of the metamodel for Eclipse's Graphical Modeling Framework (GMF).

Master_Terminal: This is the parent class for all the other terminal classes. Used in GMF for simplifying connections between terminals. Refer to the class diagram F.1 on Appendix F.

Person_Terminal: Holds the information of each person terminal that is created in the model. Required by GMF to create a shape for this class.

Medium_Terminal: Holds the information of each medium terminal that is created in the model. Required by GMF to create a shape for this class. Refer to the class diagram F.1 on Appendix F.

Connection_Terminal: Holds the information of each Connection terminal that is created in the model. Required by GMF to create a shape for this class. Refer to the class diagram F.1 on Appendix F.

Device_Terminal: Holds the information of each Device terminal that is created in the model. Required by GMF to create a shape for this class. Refer to the class diagram F.1 on Appendix F.

IsAttached_Terminal: Holds the information of each isAttached terminal that is created in the model. Required by GMF to create a shape for this class. Refer to the class diagram F.1 on Appendix F.

Capability_Terminal: Holds the information of each Capability terminal that is created in the model. Required by GMF to create a shape for this class. Refer to the class diagram F.1 on Appendix F.

Execution Subsystem: The execution subsystem includes all the classes that will be created for the handling of the execution of a model.

FileHandler: This class contains all the functionality for loading files. This contains the code for calling the file system's file open mechanisms. Refer to the class diagram F.4 on Appendix F and Appendix G for class interface.

StreamHandler: This class contains all the functionality for handling the transfer of data among the parsers. Refer to the class diagram F.4 on Appendix F and Appendix G for class interface.

MasterParser: This class is just the parent class for all the parsers. It contains functionality that is the same for all the other parsers. Refer to the class diagram F.4 on Appendix F and Appendix G for class interface.

GMLToCMLTransformer: This class handles the transformation from a GML XML file to a XCML file. It checks for the validity of the GXML and parses it to XCML. Refer to the class diagram F.4 on Appendix F and Appendix G for class interface.

SchemaTransformaer: This class handles the transformation from a schema into an instance. It also contains the functionality for prompting for the required input from the user. Refer to the class diagram F.4 on Appendix F and Appendix G for class interface.

SynthesisEngine: This class handles the creation of Operation (Commands) and binds them to SkypePtHandler. Refer to the class diagram F.5 on Appendix F and Appendix G for class interface.

SkypePtHandler: This class contains the all the Operations that have been bind for execution. Refer to the class diagram F.5 on Appendix F and Appendix G for class interface.

ConcreteOperation: This class encapsulates a Skype API call for later execution or undoing. Refer to the class diagram F.5 on Appendix F and Appendix G for class interface.

Operation: This is the interface to be implemented by ConcreteOperation. Refer to the class diagram F.5 on Appendix F and Appendix G for class interface.

SkypeOperationInvoker: This class actually executes or undoes Operations. Refer to the class diagram F.5 on Appendix F and Appendix G for class interface.

RP_AccessInterface: This is the Repository subsystem interface for the implementation of the façade pattern. Refer to the class diagram F.6 on Appendix F and Appendix G for class interface.

InformationRepository: This class handles all the information regarding the rules for the communications language. Refer to the class diagram F.6 on Appendix F and Appendix G for class interface.

5.1.2 Design Patterns

Singleton: Design pattern used to restrict instantiation of a class to one object. This design patter was chosen for the StreamHandler mainly because we did not want multiple copies of this object floating around wasting memory. Since this is our stream manager we only need one. Even though during implementation we could have decided to only have one instance of this object it was not guaranteed that it would hold especially if someone else tries to update on top of this. This design pattern was the best to make sure that no extra copies of the StreamHandler were floating around at all times.

Command: Design patter in which objects are used to represent actions. A command object encapsulates an action and its parameters. This command patterns was chosen to guarantee a

control over the execution calls to the Skype API. This way we have control when and whether o not a action gets to execute, thus preventing or canceling undesirable behavior, or even threats.

Abstract Factory: This command allow for portability. This was chosen to allow for Skype to be replaced by other platforms without the user realizing that a change has occurred. This will allow for a future Communication Virtual Machine to replace Skype with little modification to the code.

5.2 Object Interaction

5.2.1 Statechart For StreamHandler (Pipe)

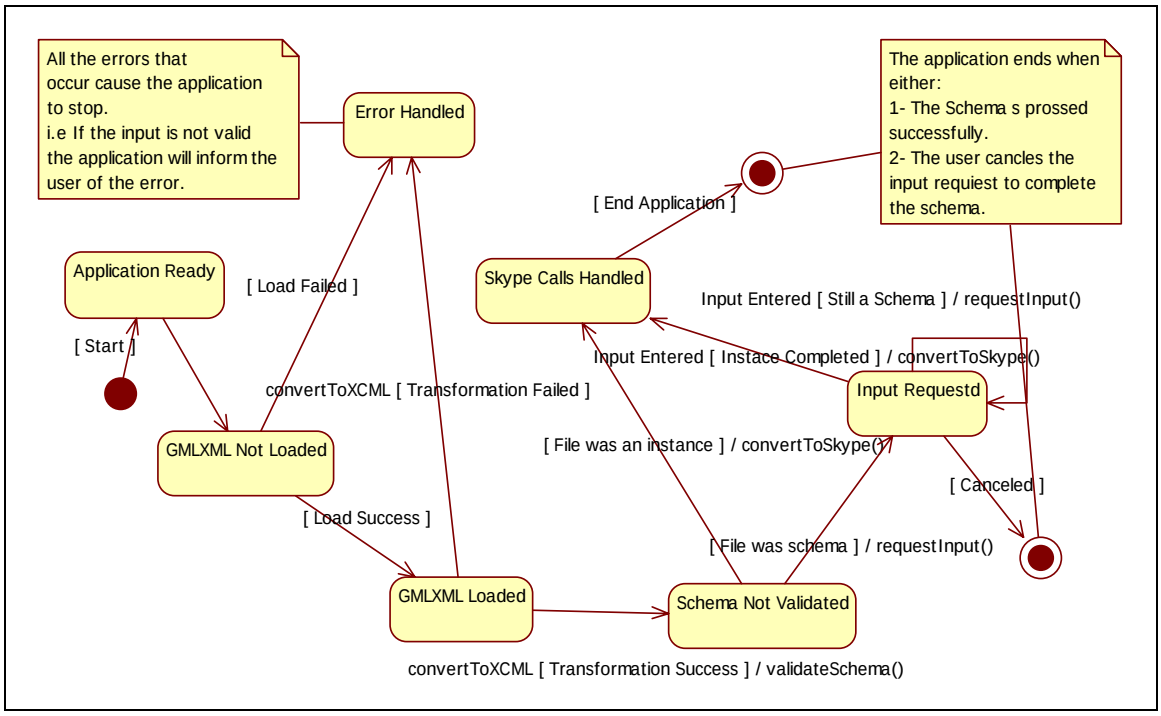


Fig 5.2.3 Statechart For StreamHandler.

5.3 Detailed Class Design

Modeling Environment Classes:

EclipesMAndITemp: This class handles the operation relevant to the menus of the application. This class handles all the templates for the domain specific menus and interfaces of the application. Refer to the class diagram F.7 on Appendix F.

EclipseVisualEditor: This class handles the interaction of the user and the modeling environment. This is the view that the user will deal with when creating models. Refer to the class diagram F.7 on Appendix F.

EclipseVisualParser: This class handles the conversion of visual input into GML files. This class parses the GML back from the GML file into shapes and connections. It also converst any changes in the **EclipseVisualEditor** into the GML file. Refer to the class diagram F.7 on Appendix F.

EclipseVisualLoader: This class handles the loading of the Communication Modeling Environment. This calls loads all the information required by the eclipse GMF to create the modeling environment. This information is loaded from the repository and it can be previously save files as well as the metamodel for the application. Refer to the class diagram F.7 on Appendix F.

EclipseVEController: This is the controller for the visual environment. This handles the interaction between the model **EclipseVisualParser** and the view **EclipseVisualEditor**. Refer to the class diagram F.7 on Appendix F.

EclipseMenuSysAndDisp: This class handles the fixed menu systems and display. This will control what menus get to be accessible in the CME. Refer to the class diagram F.7 on Appendix F.

CMEController: This is the controller of the Communication Modeling Environment. It handles the transfer of information between the models and the **EclipseVisualEditor**, as well as the **EclipseVisualLoader** and **EclipesMAndITemp.** Refer to the class diagram F.7 on Appendix F.

EclipseModelTransformer: The eclipse model transformer is that class that will handle the transformation from GML to XGML that is understood by Eclipse. This class will convert the file generated by the **EclipseVisualParser** into a file containing information more specific to the Eclipse modeling framework. Refer to the class diagram F.7 on Appendix F.

Model Creation Classes:

Master_Terminal: This is the parent class for all the other terminal classes. Used in GMF for simplifying connections between terminals. This class purpose is to create a main holder for all the terminals in GMF. Refer to the class diagram F.1 on Appendix F.

Person_Terminal: Holds the information of each person terminal that is created in the model. Required by GMF to create a shape for this class. The purpose of this class is to generate a Person shape in GMF as well as hold all the information required from a Person terminal. This information includes the person's name, the person's id, and the person's role. Refer to the class diagram F.1 on Appendix F.

Medium_Terminal: Holds the information of each medium terminal that is created in the model. Required by GMF to create a shape for this class. The purpose of this class is to generate a Medium shape in GMF as well as hold all the information required for a Medium terminal. This information includes the medium type, the suggested application, and the voice command. Refer to the class diagram F.1 on Appendix F.

Connection_Terminal: Holds the information of each connection terminal that is created in the model. Required by GMF to create a shape for this class. The purpose of this class is to generate a Connection shape in GMF as well as hold all the information required for a Connection terminal. This information includes the connection id and the bandwidth. Refer to the class diagram F.1 on Appendix F.

Device_Terminal: Holds the information of each device terminal that is created in the model. Required by GMF to create a shape for this class. The purpose of this class is to generate a Device shape in GMF as well as hold all the information required for a Device terminal. This information includes the device's id. Refer to the class diagram F.1 on Appendix F.

IsAttached_Terminal: Holds the information of each isAttached terminal that is created in the model. Required by GMF to create a shape for this class. The purpose of this class is to generate a isAttached shape in GMF as well as hold all the information required for a isAttacher terminal. This information includes the person's id and the id of the device it is attached to. Refer to the class diagram F.1 on Appendix F.

Capability_Terminal: Holds the information of each Capability terminal that is created in the model. Required by GMF to create a shape for this class. The purpose of this class is to generate a Capability shape in GMF as well as hold all the information required for a Capability terminal. This information includes the type of capability. Refer to the class diagram F.1 on Appendix F.

Execution Subsystem: The execution subsystem includes all the classes that will be created for the handling of the execution of a model.

FileHandler: This class contains all the functionality for loading files. This contains the code for calling the file system's file open mechanisms. This class purpose is to implement the instruction required for accessing the file system in the given operating system. The operation loadXMLFile(), calls the operating system's file handler and loads a file of the format .xml. Refer to the class diagram F.4 on Appendix F and Appendix G for class interface.

StreamHandler: This class contains all the functionality for handling the transfer of data among the parsers. This is the class that manages which parser gets to parse the data and when to parse it. The operation transformToXCML() calls convertToXCML() operation from GMLToCMLTransformer to transform the file into an XCML compliant file. It then calls validateSchema() which then calls validateSchema() from the SchemaTransformer to convert the schema into an instance if is not already so. Finally it calls convertToSkype() which calls the operation convertToSkypeCalls() to perform Skype API calls. Refer to the class diagram F.4 on Appendix F and Appendix G for class interface.

MasterParser: This class is just the parent class for all the parsers. It contains functionality that is the same for all the other parsers. Refer to the class diagram F.4 on Appendix F and Appendix G for class interface.

GMLToCMLTransformer: This class handles the transformation from a GML XML file to a XCML file. It checks for the validity of the GXML and parses it to XCML. The purpose of this class is to validate a the input file and convert it to the form of XCML. The operation convertToXCML() converts the file an XML file to XCML and returns the new data file to FileHandler. Refer to the class diagram F.4 on Appendix F and Appendix G for class interface.

SchemaTransformer: This class handles the transformation from a schema into an instance. It also contains the functionality for prompting for the required input from the user. The purpose of this class is to handle the transformation from a schema into an instance that is ready for execution. The operation validateSchema() takes in an XCML data file and checks if is missing the values for some of the data, if so it calls displayReuestForm() to request this information from the user. Refer to the class diagram F.4 on Appendix F and Appendix G for class interface.

SynthesisEngine: This class handles the creation of Operation (Commands) and binds them to SkypePtHandler. The purpose of this class is encapsulate the operations to be performed by Skype. The operation convertToSkypeCalls() takes in an instance data and creates Operations depending on required instruction. This is then given to SkypePtHandler by calling addSkypeOperation(). Refer to the class diagram F.5 on Appendix F and Appendix G for class interface.

SkypePtHandler: This class contains the all the Operations that have been bind for execution. This class is the client that receives all of the Operations to be executed. It will contain an Action() operation for every command that will be executed from Skype. Refer to the class diagram F.5 on Appendix F and Appendix G for class interface.

ConcreteOperation: This class encapsulates a Skype API call for later execution or undoing. This is the parent class of all the operation classes that will be created for the execution of Skype. Refer to the class diagram F.5 on Appendix F and Appendix G for class interface.

Operation: This is the interface to be implemented by ConcreteOperation. Refer to the class diagram F.5 on Appendix F and Appendix G for class interface.

SkypeOperationInvoker: This class actually executes or undoes Operations. This class the invoker of the Operation(Command) that will be executed from Skype. This is the class that will ultimately signal for the execution or undoing of the Operations. Refer to the class diagram F.5 on Appendix F and Appendix G for class interface.

RP_AccessInterface: This is the Repository subsystem interface for the implementation of the façade pattern. This interface improves the communication between subsystems. Refer to the class diagram F.6 on Appendix F and Appendix G for class interface.

InformationRepository: This class handles all the information regarding the rules for the communications language. This class also deals with the format of the schema to be parsed and the constraints of the language**.** Refer to the class diagram F.6 on Appendix F and Appendix G for class interface.

5.4 Validation of the Detailed Design Model

5.4.1 Check List

Design Architectural Model Inspection	**Yes**	**No**
Completeness: A sufficient set of interfaces is defined to provide all of the services needed for the application's functionality. The relationship between the interfaces allows for the flow of control and data and data necessary to realize all of the uses described in the use case diagram.	X	
Correctness: The architectures satisfy its constraints; use of appropriate architectural patterns; represents the interactions between the interfaces.	X	
Consistency: Each use of the system can be handled only by one set of interfaces.	X	

Syntax Checks for Classes	**Yes**	**No**
Check that multiplicity on an association is correctly represented on the class diagram.	X	
Ensure that stereotypes are represented by << >> on classes, attributes, operations and relationships on a class diagram.	X	
Check association of classes with language libraries.		X
Check to see if a class is an exception class.	X	
Check how error handling is modeled and implemented in a class.	X	

Semantic Checks for Classes	**Yes**	**No**
Check direction for association.	X	
Check the meaning of the relationships on a class diagram.	X	
Check for collection of classes.	X	
Check the business rules behind the multiplicity.	X	
Check for association classes.	X	
Check if the operations of a class that has been specialized are overloaded.	X	
Check for encapsulation.	X	
Ensure that language constructs subject to interpretations are checked for their implied meaning.	X	

Aesthetic Checks for Classes	Yes	No
Check number of attributes.	X	
Check the number of operations.	X	
Check the load on operations.	X	
Check the load on the class.	X	

Aesthetic Checks for Class Diagrams	Yes	No
Ensure that technical classes are represented only by their names rather than by their entire qualification.	X	
Improve the aesthetics by letting the entity classes appear in more than one diagram.		X
Improve the aesthetics by redistributing the classes and their associations across more than one class diagram.	X	
Ensure that sufficient explanatory note are provided.	X	
Check how error handling is modeled and implemented in a class.	X	

Syntax Check for State Chart Diagram	Yes	No
Check transitions.	X	
Check events.	X	
Check guard conditions.	X	
Check entry condition.	X	
Check exit condition.	X	
Check activity states.	X	
Check action states	X	

Semantic Check for State Chart Diagram	Yes	No
Check messages going out to other objects.	X	
Check messages being received by other objects.	X	
Check nested states.	X	
Check historical states.	X	
Check parallel states.	X	
Check to see that state chart diagrams map with objects shown for a class within a class diagram.	X	

Aesthetic Check for State Chart Diagram	Yes	No
Ensure the number of states on a diagram and their complexity is understandable.	X	

5.4.2 Structure Walkthrough

Class Diagram	Yes	No
Every class maps to an object in the sequence diagram.	X	
Operations map to transition in the sequence diagram.	X	

Class diagram represents the structure on the subsystems.	X	
Enough interfaces to facilitate communication between subsystems.	X	
Every class has a stereotype that maps to the profile.	X	

Test Case – Detailed Class Diagram	**Comments**	**PASS**	**FAIL**
TC-1.1_CrtUsrNTerm		X	
TC-1.2_CrtCnctnNTerm		X	
TC-1.3_CrtLclNTerm		X	
TC-1.4_CrtLclNTerm		X	
TC-1.6_CrtDvcNTerm		X	
TC-1.8_CrtTerm		X	
TC-1.9_SaveMdl		X	
TC-1.10_LoadMdl		X	
TC-2.1_SchTransf		X	
TC-2.2_ExecuteModel		X	
TC-1.11_CrtVoiceClCommMdl		X	
TC-1.12_CrtChatCommMdl		X	
TC-1.13_CommMdlConstcy		X	

Test Case – State Chart Diagram	**Comments**	**PASS**	**FAIL**
TC-1.1_CrtUsrNTerm		X	
TC-1.2_CrtCnctnNTerm		X	
TC-1.3_CrtLclNTerm		X	
TC-1.4_CrtLclNTerm		X	
TC-1.6_CrtDvcNTerm		X	
TC-1.8_CrtTerm		X	
TC-1.9_SaveMdl		X	
TC-1.10_LoadMdl		X	
TC-2.1_SchTransf		X	
TC-2.2_ExecuteModel		X	
TC-1.11_CrtVoiceClCommMdl		X	
TC-1.12_CrtChatCommMdl		X	
TC-1.13_CommMdlConstcy		X	

6. Implementation

This chapter introduces all the information related to the implementation of the RRCommSSys. This chapter describes the major Platform Specific Models. This models are the Eclipse Modeling Framework (EMF), the Eclipse Graphical Editing Framework (GEF), the Eclipse Graphical Modeling Framework (GMF), and the Skype platform. This chapter also describes the validation of the subsystems.

6.1 Description of the platform specific model used.

Eclipse Graphical Modeling Framework (GMF) provides a generative component and runtime infrastructure for developing graphical editors based on **EMF** and **GEF**. The project aims to provide these components, in addition to exemplary tools for select domain models which illustrate its capabilities. [4]

Graphical Editing Framework (GEF) allows developers to create a rich graphical editor from an existing application model. GEF consists of 2 plug-ins. The **org.eclipse.draw2d** plug-in provides a layout and rendering toolkit for displaying graphics. The developer can then take advantage of the many common operations provided in GEF and/or extend them for the specific domain. GEF employs an MVC (model-view-controller) architecture which enables simple changes to be applied to the model from the view. [4]

Eclipse Modeling Framework (EMF) project is a modeling framework and code generation facility for building tools and other applications based on a structured data model. From a model specification described in XMI, EMF provides tools and runtime support to produce a set of Java classes for the model, along with a set of adapter classes that enable viewing and command-based editing of the model, and a basic editor. [4]

Skype is a peer-to-peer Internet telephony network Skype has experienced rapid growth in both popular usage and software development since launch, both of its free and its paid services. The Skype communications system is notable for its broad range

of features, including free voice and video conferencing, its ability to use peer to peer (decentralized) technology to overcome common firewall and NAT (Network address translation) problems, and its extreme countermeasures against reverse engineering of the software or protocol. [3]

6.2 Validation of System

6.2.1 Check List

Syntax Checks for Classes (Advanced)	**Yes**	**No**
Check class names.	X	
Check stereotypes.	X	
Check the type of the class itself.	X	
Check attributes. It is important to subject the attributes to syntax checks that are language specific.	X	
Check attribute types.	X	
Check attribute initial values. Ensure that the type of value initialization and the attribute types are compatible.	X	
Check attribute visibility.	X	
Check attribute stereotypes.	X	
Check operations to ensure their format complies with the language of implementation.	X	
Check operation signatures.	X	
Check operation visibility.	X	
Check operation stereotypes.	X	

Semantic Checks for Classes	**Yes**	**No**
Check meaning of the class.	X	
Check meaning of the attributes.	X	
Check attribute initial values.	X	
Check What does an operation mean? Ensure that the meaning of the operation is reflected in its name and format.	X	
Check the pre- and post-conditions of operations.	X	
Check signature of operations.	X	
Check stereotypes of operations.	X	
Check scope of operations.	X	
Check to see if the operations of a class are overloaded.	X	
If overriding operations exist, ensure their correctness.	X	
Check for overriding variables.	X	
Check for encapsulation.	X	

6.2.2 Implementation Test

Identifier:	TC-1.1_CrtUsrNTerm
Expected results:	The Connection is drawn onto the canvas.
Actual results:	The Connection is drawn onto the canvas.
Pass/Fail:	**PASS**

Identifier:	TC-1.2_CrtCnctnNTerm
Expected results:	The Connection terminal is drawn on the canvas. The connections are drawn between the Connection terminal and the MediaAttached terminal.
Actual results:	The Connection terminal is drawn on the canvas. The connections are drawn between the Connection terminal and the MediaAttached terminal.
Pass/Fail:	**PASS**

Identifier:	TC-1.3_CrtLclNTerm
Expected results:	The Person terminal is drawn on the canvas. The isAttached terminal is drawn on the canvas. The connections are drawn between the Person terminal and the isAttached terminal.
Actual results:	The Person terminal is drawn on the canvas. The isAttached terminal is drawn on the canvas. The connections are drawn between the Person terminal and the isAttached terminal.
Pass/Fail:	**PASS**

Identifier:	TC-1.4_CrtLclNTerm
Expected results:	The Person terminal is drawn on the canvas. The isAttached terminal is drawn on the canvas. The connections are drawn between the Person terminal and the isAttached terminal.
Actual results:	The Person terminal is drawn on the canvas. The isAttached terminal is drawn on the canvas. The connections are drawn between the Person terminal and the isAttached terminal.
Pass/Fail:	**PASS**

Identifier:	TC-1.5_CrtMdaAtchNTerm
Expected results:	The Medium shape is drawn on the canvas, as well as the connection to the Form non-terminal.

Actual results:	The Medium shape is drawn on the canvas, as well as the connection to the Form non-terminal.
Pass/Fail:	**PASS**

Identifier:	TC-1.6_CrtDvcNTerm
Expected results:	The Capability shape and the Device shape are drawn on the canvas, as well as the connection from Device to Capability.
Actual results:	The Capability shape and the Device shape are drawn on the canvas, as well as the connection from Device to Capability.
Pass/Fail:	**PASS**

Identifier:	TC-1.8_CrtTerm
Expected results:	The Terminal shape is drawn on the canvas.
Actual results:	The Terminal shape is drawn on the canvas.
Pass/Fail:	**PASS**

Identifier:	TC-1.9_SaveMdl
Expected results:	The file is saved.
Actual results:	The file is saved.
Pass/Fail:	**PASS**

Identifier:	TC-1.10_LoadMdl
Expected results:	The file is loaded.
Actual results:	The file is loaded.
Pass/Fail:	**PASS**

Identifier:	TC-2.1_SchTransf
Expected results:	The schema is transformed into am instance.
Actual results:	The schema is transformed into am instance.
Pass/Fail:	**PASS**

Identifier:	TC-2.2_ExecuteModel
Expected results:	The instance is executed and Skype calls are made.
Actual results:	The instance is executed and Skype calls are made.
Pass/Fail:	**PASS**

Identifier:	TC-1.11_CrtVoiceClCommMdl
Expected results:	All the terminals and connections are drawn on the canvas.
Actual results:	All the terminals and connections are drawn on the canvas.
Pass/Fail:	**PASS**

Identifier:	TC-1.12_CrtChatCommMdl
Expected results:	All the terminals and connections are drawn on the canvas.
Actual results:	All the terminals and connections are drawn on the canvas.
Pass/Fail:	**PASS**

7. Glossary

Class Diagram – A model representing the different classes within a s/w 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.

Postcondition – 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.

RRCommSSys – Name of this system – Rapid Realization of communication Services System.

VE/VDE - Visual Environment/Visual Development Environment**.** Interface where the user can develop communication models by selecting terminals from a toolbox and dragging them onto the canvas.

GEF - Graphical Editing Framework. The Graphical Editing Framework is an open source framework dedicated to providing a rich, consistent graphical editing environment for
applications on the Eclipse Platform.

GMF - Graphical Modeling Framework. Creates a modeling framework which we used to create the VE.

CVM - Communication Virtual Machine. Communication Virtual machine.

CML - Communication Modeling Language. Language establishing the constraints for communication models.

EMF – Eclipse Modeling Framework. Used with GMF and GEF to build out VE.

8. Appendix

8.1 Appendix A – Use Case Diagrams

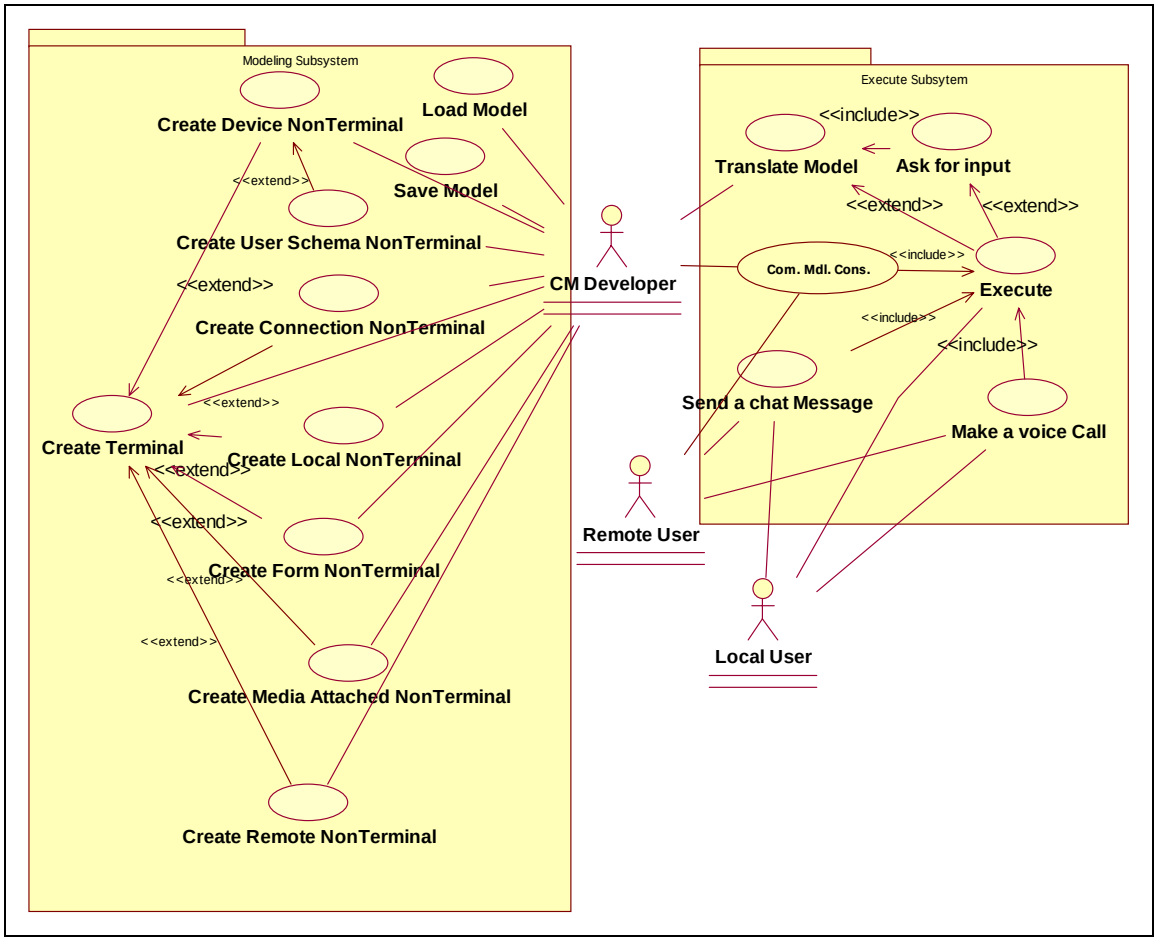


Fig A.1 Use Case Diagram for RRComSys.

8.2 Appendix B – Use Cases

Use Case – Create User Schema Non-Terminal

***Use Case ID:** 1.1_CrtUsrNTerm

Use Case Level: High-level.

***Scenario:** User creates a User Schema non-terminal from existing terminals and nonterminals

- **Actor:** Devoloper for Modeling environment.

  Pre-conditions: Modeling environment has to be loaded. Terminals and nonTerminals for a Local non-terminal(see use case 1.3_CrtLclNTerm) and Connections Non-terminals (see use case 1.2_CrtCnctnNTerm) must be on the canvas.

- **Description:**

  1. Trigger: The user drags the connection to Local connection, connecting the local and connection non terminals.
  2. The system responds by drawing the line between the two non-terminals

- **Post-conditions:** The Model in the modeling canvas now contains a User Schema Non-Terminal

***Alternative Courses of Action:** User can connect more more than one connection to the local non terminal. The system shall draw the lines accordingly.

Extensions: N/A

***Exceptions:** The user might try to connect the wrong association to the non-terminals. In this case, the system will not draw the line, as the validity of the model is checked at run-time.

The user might try to connect the right association but to the wrong type of terminals. The system will not draw the line for the same reason.

Concurrent Uses: N/A

***Related Use Cases:** 1.3_CrtLclNTerm, 1.2_CrtCnctnNTerm

Decision Support

***Frequency:** Average 3 per model, 20 per day.

***Criticality:** Intermediate

***Risk:** Medium.

***Constraints:**

- Usability: *must be easy to use, can be mastered in less than 5 minutes.*
- Reliability: *must always draw the selected terminal or connection.*
- Performance: *must take no longer than 2 seconds to draw the terminal shape or connection selected.*
- Supportability: - No debug necessary.
- Implementation: *must be implemented with Eclipse GMF*.

Modification History --

***Owner:** Alejandro Ortiz

***Initiation date:** 02/05/07

***Date last modified:** 02/05/07

Use Case – Create Connection Non-Terminal

***Use Case ID:** 1.2_CrtCnctnNTerm

Use Case Level: High-level.

***Scenario:** User creates a Connection non-terminal from existing terminals and nonterminals

- **Actor:** Devoloper for Modeling environment.

  Pre-conditions: Modeling environment has to be loaded. NonTerminals for a Media Attached non-terminal(see use case 1.5_CrtMdiaAttchNTerm), and one or more Remote Non-terminals (see use case 1.4_CrtRmtNTerm) and must be on the canvas.

- **Description:**

1. Trigger: User drags a Connections terminal onto the canvas. (see use case 1.8_CrtTerm). System responds as that usecase indicates.
2. Trigger: The user drags the Connection to media Attached connection, connecting the Media Attached non-terminal and connection terminal.
3. The system responds by drawing the line between the non-terminal and the terminal shape.
4. Trigger: User then drags the connection to Remote connection, connecting connection terminal and remote non-terminal.

5. The system responds by drawing the line between the non-terminal and the terminal shape.

- **Post-conditions:** The Model in the modeling canvas now contains a connection Non-Terminal

***Alternative Courses of Action:** User can connect more than one remotes to the connection terminal. The system shall draw the lines accordingly.

Extensions: N/A

***Exceptions:** The user might try to connect the wrong association to the non-terminals. In this case, the system will not draw the line, as the validity of the model is checked at run-time.

The user might try to connect the right association but to the wrong type of terminals. The system will not draw the line for the same reason.

Concurrent Uses: 1.8_CrtTerm

***Related Use Cases:** 1.5_ CrtMdiaAttchNTerm, 1.4_ CrtRmtNTerm

---

Decision Support

***Frequency:** Average 3 per model, 20 per day.

***Criticality:** Intermediate

***Risk:** Medium.

***Constraints:**

- Usability: *must be easy to use*
- Reliability: *must always draw the selected terminal or connection.*
- Performance: *must take no longer than 2 seconds to draw the terminal shape or connection selected.*
- Supportability: - No debug necessary.
- Implementation: *must be implemented with Eclipse GMF.*

Modification History --

***Owner:** Alejandro Ortiz

***Initiation date:** 02/05/07

***Date last modified:** 02/05/07

Use Case – Create Local Non-Terminal

***Use Case ID:** 1.3_CrtLclNTerm

Use Case Level: High-level.

Scenario: User creates a Local non-terminal from existing terminals and nonterminals

- **Actor:** Devoloper for Modeling environment.
- **Pre-conditions:** Modeling environment has to be loaded. nonTerminals for a device non-terminal(see use case 1.6_CrtDvcNTerm), must be on the canvas.
- **Description:**

1. Trigger: User drags a person terminal onto the canvas. (see use case 1.8_CrtTerm). System responds as that usecase indicates.
2. Trigger: User drags a isAttached terminal onto the canvas. (see use case 1.8_CrtTerm). System responds as that usecase indicates.
3. Trigger: The User drags the person to is attached connection, connecting the person and is attached terminals.
4. The system responds by drawing the line between the two terminals.
5. Trigger: the user then drags an is attached to device connection, connecting the is attached terminal and the device nonterminal.
6. The system responds by drawing the line between the terminal and non-terminal.

- **Post-conditions:** The Model in the modeling canvas now contains a User Schema Non-Terminal

***Alternative Courses of Action:** N/A

Extensions: N/A

***Exceptions:** The user might try to connect the wrong association to the non-terminals. In this case, the system will not draw the line, as the validity of the model is checked at run-time.

The user might try to connect the right association but to the wrong type of terminals. The system will not draw the line for the same reason.

Concurrent Uses: 1.8_CrtTerm

***Related Use Cases:** 1.6_ CrtDvcNTerm

Decision Support

***Frequency:** Average 3 per model, 20 per day.

***Criticality:** Intermediate

***Risk:** Medium.

***Constraints:**

- Usability: *must be easy to use, can be mastered in less than 5 minutes.*
- Reliability: *must always draw the selected terminal or connection.*
- Performance: *must take no longer than 2 seconds to draw the terminal shape or connection selected.*
- Supportability: - No debug necessary.
- Implementation: *must be implemented with Eclipse GMF*.

Modification History --

***Owner:** Alejandro Ortiz

***Initiation date:** 02/05/07

***Date last modified:** 02/05/07

Use Case – Create Remote Non-Terminal

***Use Case ID:** 1.4_CrtRmtNTerm

Use Case Level: High-level.

*** Scenario:** User creates a Remote non-terminal from existing terminals and nonterminals

- **Actor:** Devoloper for Modeling environment.
- **Pre-conditions:** Modeling environment has to be loaded. Terminals and nonTerminals for a device non-terminal(see use case 1.6_CrtDvcNTerm), a person terminal, and an isAttached terminal (see use case 1.8_CrtTerm) must be on the canvas.
- **Description:**
    1. Trigger: User drags a person terminal onto the canvas. (see use case 1.8_CrtTerm). System responds as that usecase indicates.
    2. Trigger: User drags a isAttached terminal onto the canvas. (see use case 1.8_CrtTerm). System responds as that usecase indicates.
    3. Trigger: The User drags the person to is attached connection, connecting the person and is attached terminals.
    4. The system responds by drawing the line between the two terminals.

5. Trigger: the user then drags an is attached to device connection, connecting the is attached terminal and the device nonterminal.
6. The system responds by drawing the line between the terminal and non-terminal.

- **Post-conditions:** The Model in the modeling canvas now contains a Remote Non-Terminal

***Alternative Courses of Action:** N/A

Extensions: N/A

***Exceptions:** The user might try to connect the wrong association to the non-terminals. In this case, the system will not draw the line, as the validity of the model is checked at run-time.

The user might try to connect the right association but to the wrong type of terminals. The system will not draw the line for the same reason.

Concurrent Uses: 1.8_CrtTerm

***Related Use Cases:** 1.6_ CrtDvcNTerm

Decision Support

***Frequency:** Average 3 per model, 20 per day.

***Criticality:** Intermediate

***Risk:** Medium.

***Constraints:**

- Usability: *must be easy to use, can be mastered in less than 5 minutes.*
- Reliability: *must always draw the selected terminal or connection.*
- Performance: *must take no longer than 2 seconds to draw the terminal shape or connection selected.*
- Supportability: - No debug necessary.
- Implementation: *must be implemented with Eclipse GMF*.

Modification History --

***Owner:** Alejandro Ortiz

***Initiation date:** 02/05/07

***Date last modified:** 02/05/07

Use Case – Create MediaAttached Non-Terminal

***Use Case ID:** 1.5_CrtMdaAtchNTerm

Use Case Level: High-level.

*** Scenario:** User creates a Remote non-terminal from existing terminals and nonterminals

- **Actor:** Devoloper for Modeling environment.
- **Pre-conditions:** Modeling environment has to be loaded. nonTerminals for one or more form non-terminal (see use case 1.7_CrtFormNTerm) must be on the canvas.
- **Description:**
    1. Trigger: User drags a Medium terminal onto the canvas. (see use case 1.8_CrtTerm). System responds as that usecase indicates.
    2. Trigger: The user drags the form to medium connection, connecting the Medium terminal and form non terminals.
    3. The system responds by drawing the line between the two non-terminals
- **Post-conditions:** The Model in the modeling canvas now contains a MediaAttached Non-Terminal

***Alternative Courses of Action:** User can connect more than one form to the Medium terminal. The system shall draw the lines accordingly. Similarly, the user can connect more than one Medium to the form non-terminal. The system shall draw the lines accordingly.

Extensions: N/A

***Exceptions:** The user might try to connect the wrong association to the non-terminals. In this case, the system will not draw the line, as the validity of the model is checked at run-time.

The user might try to connect the right association but to the wrong type of terminals. The system will not draw the line for the same reason.

Concurrent Uses: 1.8_CrtTerm

***Related Use Cases:** 1.7_CrtFormNTerm

Decision Support

***Frequency:** Average 3 per model, 20 per day.

***Criticality:** Intermediate

***Risk:** Medium.

***Constraints:**

- Usability: *must be easy to use, can be mastered in less than 5 minutes.*
- Reliability: *must always draw the selected terminal or connection.*
- Performance: *must take no longer than 2 seconds to draw the terminal shape or connection selected.*
- Supportability: - No debug necessary.
- Implementation: *must be implemented with Eclipse GMF*.

Modification History --

***Owner:** Alejandro Ortiz

***Initiation date:** 02/05/07

***Date last modified:** 02/05/07

Use Case – Create Device Non-Terminal

***Use Case ID:** 1.6 CrtDvcNTerm

Use Case Level: High-level.

*** Scenario:** User creates a Device non-terminal from existing terminals and nonterminals

- **Actor:** Devoloper for Modeling environment.
- **Pre-conditions:** Modeling environment has to be loaded.
- **Description:**
    1. Trigger: User drags a Capability terminal onto the canvas. (see use case 1.8_CrtTerm). System responds as that usecase indicates.
    2. Trigger: User drags a Device terminal onto the canvas. (see use case 1.8_CrtTerm). System responds as that usecase indicates.
    3. Trigger: The user drags the Device to Capability connection, connecting the Capability and Device terminals.
    4. The system responds by drawing the line between the two non-terminals
- **Post-conditions:** The Model in the modeling canvas now contains a Device Non-Terminal

***Alternative Courses of Action:** User can connect more than one Capability to the Device terminal. The system shall draw the lines accordingly.

Extensions: N/A

***Exceptions:** The user might try to connect the wrong association to the non-terminals. In this case, the system will not draw the line, as the validity of the model is checked at run-time.

The user might try to connect the right association but to the wrong type of terminals. The system will not draw the line for the same reason.

Concurrent Uses: 1.8_CrtTerm

***Related Use Cases:** N/A

Decision Support

***Frequency:** Average 3 per model, 20 per day.

***Criticality:** Intermediate

***Risk:** Medium.

***Constraints:**

- Usability: *must be easy to use, can be mastered in less than 5 minutes.*
- Reliability: *must always draw the selected terminal or connection.*
- Performance: *must take no longer than 2 seconds to draw the terminal shape or connection selected.*
- Supportability: - No debug necessary.
- Implementation: *must be implemented with Eclipse GMF*.

Modification History --

***Owner:** Alejandro Ortiz

***Initiation date:** 02/05/07

***Date last modified:** 02/05/07

Use Case – Create Form Non-Terminal

***Use Case ID:** 1.7_CrtFormNTerm

Use Case Level: High-level.

*** Scenario:** User creates a Form non-terminal from existing terminals and nonterminals

- **Actor:** Devoloper for Modeling environment.
- **Pre-conditions:** Modeling environment has to be loaded.
- **Description:**
    1. Trigger: User drags a Form terminal onto the canvas. (see use case 1.8_CrtTerm). System responds as that usecase indicates.
- **Post-conditions:** The Model in the modeling canvas now contains a User Schema Non-Terminal

***Alternative Courses of Action:** User can create a Form non-terminal also by placing one or more existing forms terminals, and one or more existing Medium terminals, into another Form terminal. The result will be a Form non-terminal containing one or more forms and/or one or more mediums.

Extensions: N/A

***Exceptions:** The user might try to connect the wrong association to the non-terminals. In this case, the system will not draw the line, as the validity of the model is checked at run-time.

The user might try to connect the right association but to the wrong type of terminals. The system will not draw the line for the same reason.

Concurrent Uses: 1.8_CrtTerm

***Related Use Cases:** N/A

Decision Support

***Frequency:** Average 3 per model, 20 per day.

***Criticality:** Intermediate

***Risk:** Medium.

***Constraints:**

- Usability: *must be easy to use, can be mastered in less than 5 minutes.*
- Reliability: *must always draw the selected terminal or connection.*
- Performance: *must take no longer than 2 seconds to draw the terminal shape or connection selected.*

- Supportability: - No debug necessary.
- Implementation: *must be implemented with Eclipse GMF*.

Modification History --

***Owner:** Alejandro Ortiz

***Initiation date:** 02/05/07

***Date last modified:** 02/05/07

Use Case – Create Terminal

***Use Case ID:** 1.8_CrtTerm

Use Case Level: *High-level.*

***Scenario:** *The CVM developer puts a terminal shape on the canvas.*

- **Actor:** *CVM Developer.*
- **Pre-conditions:** *A CVM model canvas must exist in the Eclipse IDE.*
- **Description:**
  1. The CVM developer clicks the shape on the shape palette that he or she wants to include in the CVM model.
  2. The system highlights the shape on the palette.
  3. While holding the mouse button pressed, the CVM developer drags the shape onto the canvas and releases the mouse button.
  4. The system draws the selected shape on the canvas and assigns default values for its attributes.
- **Relevant requirements:** *None.*
- **Post-conditions:** *The terminal shape (Connection, IsAttachedTo, Device, Person, Medium, Form, Capability) chose by the CVM developer is drawn on the canvas.*

***Alternative Courses of Action:** *None*

Extensions: *None*

***Exceptions:** *If the terminal shape is dropped on a place beyond the canvas boundaries, the system will not draw the shape and the mouse cursor will go back to normal.*

Concurrent Uses: *None*

***Related Use Cases:** *This use case is extended by the family of use cases related to building non-terminal constructs of the CVM modeling language.*

Decision Support

***Frequency:** *Typically, this is use case is executed every time the CVM developer wants to add non-terminal productions of the CVM language to his or her model. Non-terminals are made up of terminal shapes and relationships among, which generally results in a composite production.*

***Criticality:** *This is elementary piece of funcionality in order for the non-terminals to be composed in any model.*

***Risk:** *Without this function, the CVM developer will not be able to construct any model.*

***Constraints:**

- Usability: *must be easy to use, can be mastered in less than 5 minutes.*
- Reliability: *must always draw the selected terminal.*
- Performance: *must take no longer than 1 second to draw the terminal shape.*
- Supportability: - No debug necessary.
- Implementation: *must be implemented with Eclipse GMF*.

Modification History

***Owner:** Ariel Cary

***Initiation date:** 02/11/2007

***Date last modified:** 02/12/2007

Version: 1.0

Use Case – Save Model

***Use Case ID:** 1.9_SaveMdl.

Use Case Level: *High-level.*

***Scenario:** *The CVM developer saves the modifications done on the current CVM Model.*

- **Actor:** *CVM Developer.*
- **Pre-conditions:**
  - *A model canvas must exist containing at least one terminal shape.*
- **Description:**
  1. The CVM developer selects the Save option from the File Menu of the Eclipse IDE.

2. The system asks the CVM developer for a file name in case this model was not saved before. Otherwise, go to step 4.
3. The CVM developer enters a file name for the CVM model.
4. The system makes the model changes persistent with the file name provided in step 3 or existing file identified in step 2. The file is saved in any directory of the file system chosen by the CVM developer.
5. The system acknowledges the operation by displaying a message on the status bar of the IDE.

- **Relevant requirements:** *None.*
- **Post-conditions:** *The CVM model is saved on disk.*

***Alternative Courses of Action:** *None*

1. If in step 2, the CVM developer hits the *Cancel* button, the system cancels the action and closes the save dialog window.

Extensions: *None*

***Exceptions:**

- In step 3, if the file name provided by the CVM developer already exists in location specified by the CVM developer, then the system displays an error message accordingly and goes to step 2 to ask for another file name.
- In step 4, if there is not enough space on the file system, the system displays an error dialog message stating so.

Concurrent Uses: *None*

***Related Use Cases:** *None*

Decision Support

***Frequency:** *Typically, the CVM developer will save the current model every time a relevant change to it is made.*

***Criticality:** *This is a critical funcionality as it implements the persistency of the model so as to prevent the lost of work done on the model in case of, for exampler, a power outage or operating system error.*

***Risk:** *Fairlure to execute this function may result in lost of CVM model data.*

***Constraints:**

- Usability: *must be easy to use, can be mastered in less than 5 minutes.*
- Reliability: *must always save the CVM model on disk unless external conditions prevent doing so.*
- Performance: *is proporcional to the size of the model. It's expected to perform within 2 seconds on average.*

- Supportability: *must clearly identify the subcomponent where an error occurred in case of exceptions.*
- Implementation: *must be implemented with Eclipse GMF.*

Modification History

***Owner:** Ariel Cary

***Initiation date:** 02/11/2007

***Date last modified:** 02/14/2007

Version: 1.0

Use Case – Load Model

***Use Case ID:** 1.10_LoadMdl

Use Case Level: *High-level.*

***Scenario:** *The CVM developer loads a CVM Model into the canvas.*

- **Actor:** *CVM Developer.*
- **Pre-conditions:** *None*
- **Description:**
    1. The CVM developer selects the *Open* option from the File Menu of the Eclipse IDE.
    2. The system asks the CVM developer for the file name containing a CVM model to be loaded into a canvas.
    3. The CVM developer enters the requested file name.
    4. The system opens the file specified in step 3 and loads the CVM model into the IDE canvas.
    5. The system acknowledges the operation by displaying a message on the status bar of the IDE.
- **Relevant requirements:** *None.*
- **Post-conditions:** *The CVM model is loaded into the canvas.*

***Alternative Courses of Action:**

2. If in step 2, the CVM developer hits the *Cancel* button, the system cancels the action and closes the *Load* dialog window.

Extensions: *None*

***Exceptions:**

- In step 3, if the file name specified by the CVM developer does not exist, then the system displays an error message accordingly and goes to step 2 to ask for another file name.
- In step 4, if there is not enough space on the file system, the system displays an error dialog message stating so.
- In step 4, if the specified file does not contain a valid model or if the file cannot be read, the system displays an error dialog message stating so.

Concurrent Uses: *None*

***Related Use Cases:** *None*

Decision Support

***Frequency:** *None.*

***Criticality:** *This funcionality provides a way to re-load stored CVM models for the purpose of visualizing, modifying, etc. them.*

***Risk:** *Fairlure to execute this function will prevent CVM developers from visualizing saved models.*

***Constraints:**

- Usability: *must be easy to use, can be mastered in less than 5 minutes.*
- Reliability: *must always load the CVM model from disk unless external conditions prevent doing so.*
- Performance: *is proporcional to the size of the model. It's expected to perform within 2 seconds on average.*
- Supportability: *must clearly identify the subcomponent where an error occurred in case of exceptions.*
- Implementation: *must be implemented with Eclipse GMF.*

Modification History

***Owner:** Ariel Cary

***Initiation date:** 02/11/2007

***Date last modified:** 02/15/2007

Version: 1.0

Use Case – Schema Transformation

***Use Case ID:** *2.1_SchTransf*

Use Case Level: *System Level*

***Scenario** User John Doe wants to convert a schema of a model into an instance ready for execution. The user loads the schema into the Schema Transformation Environment (STE). He then fills in the values requested from the STE and clicks finish, and a complete instance of the schema is created and saved.

- **Actor:** *John Doe*
- **Pre-conditions:** *The input must be an incomplete schema.*
- **Description:**

    1. **Trigger:** The user initiates an action by selecting the schema file.
    2. The system responds by parsing the file and asking for missing fields

- **Relevant requirements:** *None*
- **Post-conditions:** *A complete X-CML instance is created.*

***Alternative Courses of Action** *None*

Extensions: *None*

***Exceptions:** *None*

Concurrent Uses: *None*

***Related Use Cases:** *2.2_ExecuteModel*

Decision Support

***Frequency:** *Eveytime a schema is loaded for execution.*

***Criticality:** *Critical, without it no shcemas will be allowed for execution. Only complete istances will be allowed.*

***Risk:** *User inputs wrong values, or malicious values.*

***Constraints:**

A schema is but an incomplete instance. It is an instance where at least one required field has been left empty.

- Usability: *must be easy to use, can be mastered in less than 5 minutes.*
- Reliability: *must always load the shcema from disk unless external conditions prevent doing so.*

- Performance: *is proporcional to the size of the schema. It's expected to perform within 2 seconds on average.*
- Supportability: *must clearly identify the subcomponent where an error occurred in case of exceptions.*
- Implementation: *must be implemented with Eclipse.*

Modification History – v1

***Owner:** Frank Hernandez

***Initiation date:** 02/08/2007

***Date last modified:** 02/08/2007

Note the sections with the *** must be included in the use case**.

Thanks to Dr. John McGregor, Computer Science Department Clemson University.

Use Case – Execute Model

***Use Case ID:** *2.2_ExecuteModel*

Use Case Level: *System Level*

***Scenario:** User John Doe wants to execute a communication instance. The user then loads the instance into the Synthesis Engine (SE) and selects execute.

- **Actor:** *John Doe*
- **Pre-conditions:** *The input must be an compete instance.*
- **Description:**
    1. **Trigger:** The user initiates an action by selecting the instance file.

       The system responds by parsing the file and performing the required Skype API calls.
- **Post-conditions:** *A communication model is executed.*

***Alternative Courses of Action** *None*

Extensions: *None*

***Exceptions:** *None*

Concurrent Uses: *None*

***Related Use Cases:** *2.1_SchTransf.*

Decision Support

***Frequency:** *Eveytime an instance is loaded for execution.*

***Criticality:** *Critical, without it no execution will be allowed.*

***Risk:** *None.*

***Constraints:**

An instance is a complete schema.

- Usability: *must be easy to use, can be mastered in less than 5 minutes.*
- Reliability: *must always load the instance from disk unless external conditions prevent doing so.*
- Performance: *is proporcional to the size of the schema. It's expected to perform within 2 seconds on average.*
- Supportability: *must clearly identify the subcomponent where an error occurred in case of exceptions.*
- Implementation: *must be implemented with Eclipse.*

Modification History – *v1*

***Owner:** Frank Hernandez

***Initiation date:** 02/08/2007

***Date last modified:** 02/08/2007

Use Case – Create Voice Call Communication Model

***Use Case ID:** *1.11_CrtVoiceClCommMdl*

Use Case Level: *High Level*

***Scenario:** Actor John Doe wants to create a communication model to chat with Jane Doe. He then proceeds to select and drop the terminals required for the connection to be created.

- **Actor:** *John Doe*

- **Pre-conditions:** *There must be a new file already opened and ready to be worked on.*
- **Description:**

Trigger: The user initiates an action by draging a terminal onto the screen.

The system responds by painting the terminal onto the canvas

1. Actor drags the termial for ‘Person’ onto the canvas and fills in the required information.
2. Actor drags the termial for ‘isAttahced’ onto the canvas and fills in the required information, and attaches it to ‘Person’.
3. Actor drags the termial for ‘Device’ onto the canvas and fills in the required information, and attaches it to ‘isAttached’.
4. Repeats step 1-3 for Jane Doe.
5. Actor drags the termial for ‘Connnection’ onto the canvas and fills in the required information, and connects it to both ‘Device’ terminals he created previusly .
6. Actor drags the termial for ‘Medium’ onto the canvas and fills in ‘LiveAudio’, and connects to the ‘Connection’ terminal he created.

- **Post-conditions:** *After the use the system will now hold the a communications model that stablishes a voice call connection between 2 actors.*

***Alternative Courses of Action** *There can be a conference call connection model, this will be similar to this main use with the exception that rather than having a set of terminal created for Jane Doe, now we have similar sets of terminal created for each actor that will be part of the conference.*

Extensions: *Team1_Conference_ Call.*

***Exceptions:** *No errors araise from this use case.*

Concurrent Uses: *1.11_CrtChatCommMdl*

***Related Use Cases:** *N/A*

Decision Support

***Frequency:** *This use will be persormed at least once per application use. That is everytime that that any actor want to create a communications model to connect to another actor.*

***Criticality:** *This use is **required** for anytime the user will want to create a voice call model to communicate to another actor.*

***Risk:** *The is little or no risk involded in this use.*

***Constraints:**

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

- Usability: *must be easy to use, can be mastered in less than 5 minutes.*
- Reliability: *must always draw the selected terminal or connection.*
- Performance: *must take no longer than 2 seconds to draw the terminal shape or connection selected.*
- Supportability: - No debug nescesary, simple.
- Implementation: *must be implemented with Eclipse GMF*.

Modification History – *v1*

***Owner:** Frank Hernandez

***Initiation date:** 02/08/2007

***Date last modified:** 02/08/2007

Use Case – Create Chat Message Model

***Use Case ID:** *1.12_CrtChatCommMdl*

Use Case Level: *High Level*

***Scenario:** Actor John Doe wants to create a communication model to chat with Jane Doe. He then proceeds to select and drop the terminals required for the connection to be created.

- **Actor:** *John Doe*
- **Pre-conditions:** *There must be a new file already opened and ready to be worked on.*
- **Description:**

  Trigger: The user initiates an action by draging a terminal onto the screen.

  The system responds by painting the terminal onto the canvas

  1. Actor drags the termial for ‘Person’ onto the canvas and fills in the required information.
  2. Actor drags the termial for ‘isAttahced’ onto the canvas and fills in the required information, and attaches it to ‘Person’.
  3. Actor drags the termial for ‘Device’ onto the canvas and fills in the required information, and attaches it to ‘isAttached’.
  4. Repeats step 1-3 for Jane Doe.

5. Actor drags the termial for ‘Connnection’ onto the canvas and fills in the required information, and connects it to both ‘Device’ terminals he created previusly .
6. Actor drags the termial for ‘Medium’ onto the canvas and fills in ‘TextFile’, and connects to the ‘Connection’ terminal he created.

- **Relevant requirements:** *None*
- **Post-conditions:** *After the use the system will now hold the a communications model that stablishes a voice call connection between 2 actors.*

***Alternative Courses of Action** *There can be a conference chat connection model, this will be similar to this main use with the exception that rather than having a set of terminal created for Jane Doe, now we have similar sets of terminal created for each actor that will be part of the conference.*

Extensions: *Team1_Conference_ Chat.*

***Exceptions:** *No errors araise from this use case.*

Concurrent Uses: *Team1_Voice_Call*

***Related Use Cases:** *N/A*

Decision Support

***Frequency:** *This use will be persormed at least once per application use. That is everytime that that any actor want to create a communications model to connect to another actor.*

***Criticality:** *This use is **required** for anytime the user will want to create a chat model to communicate to another actor.*

***Risk:** *The is little or no risk involded in this use.*

***Constraints:**

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

- Usability: *must be easy to use, can be mastered in less than 5 minutes.*
- Reliability: *must always draw the selected terminal or connection.*
- Performance: *must take no longer than 2 seconds to draw the terminal shape or connection selected.*
- Supportability: - No debug nescesary, simple.
- Implementation: *must be implemented with Eclipse GMF.*

Modification History – *v1*

***Owner:** Frank Hernandez

***Initiation date:** 02/08/2007

***Date last modified:** 02/08/2007

Use Case – Communication Model Consistency

***Use Case ID:** *1.13_CommMdlConstcy*

Use Case Level: *High-level.*

***Scenario:** *A misuser tries to execute a communication model including the same Skype username more than once.*

- **Actor:** *CVM Misuser.*
- **Pre-conditions:**
  - *A communication model previously developed in the Communication Environment must exist.*
- **Description:**
  1. The misuser tries to execute the model, that is to instanciate the communication schema.
  2. If the schema does not contain persons already set, the system asks the miuser to enter them.
  3. The misuser enters the same Skype username (person) for either the local and remote connections or for more than one remote connection.
  4. The system validates the communication instance and encounters the invalid scenario.
  5. The system notifies the error condition to the misuser and does not execute the communication instance.
- **Relevant requirements:** *None.*
- **Post-conditions:** *The system does not execute the communication model.*

***Alternative Courses of Action:** *None*

Extensions: *None*

***Exceptions:** None

Concurrent Uses: *None*

***Related Use Cases:** *None*

Decision Support

***Frequency:** *This threat can potentially occur anytime when the CVM user executes a communication model.*

***Criticality:** *This is a critical integrity use case as it prevents the same person to be bound more than once in a communication model, which could result in a inconsistent scenario.*

***Risk:** *Fairlure to execute this function may result in an inconsistent communication model.*

***Constraints:**

- Usability: -
- Reliability: *the system must always validate this type of error condition.*
- Performance: -
- Supportability: *must clearly identify the subcomponent where an error occurred in case of exceptions.*
- Implementation: -

Modification History

***Owner:** Ariel Cary

***Initiation date:** 02/11/2007

***Date last modified:** 03/20/2007

Version: 1.0

8.3 Appendix C – Class Diagram For Analysis Model

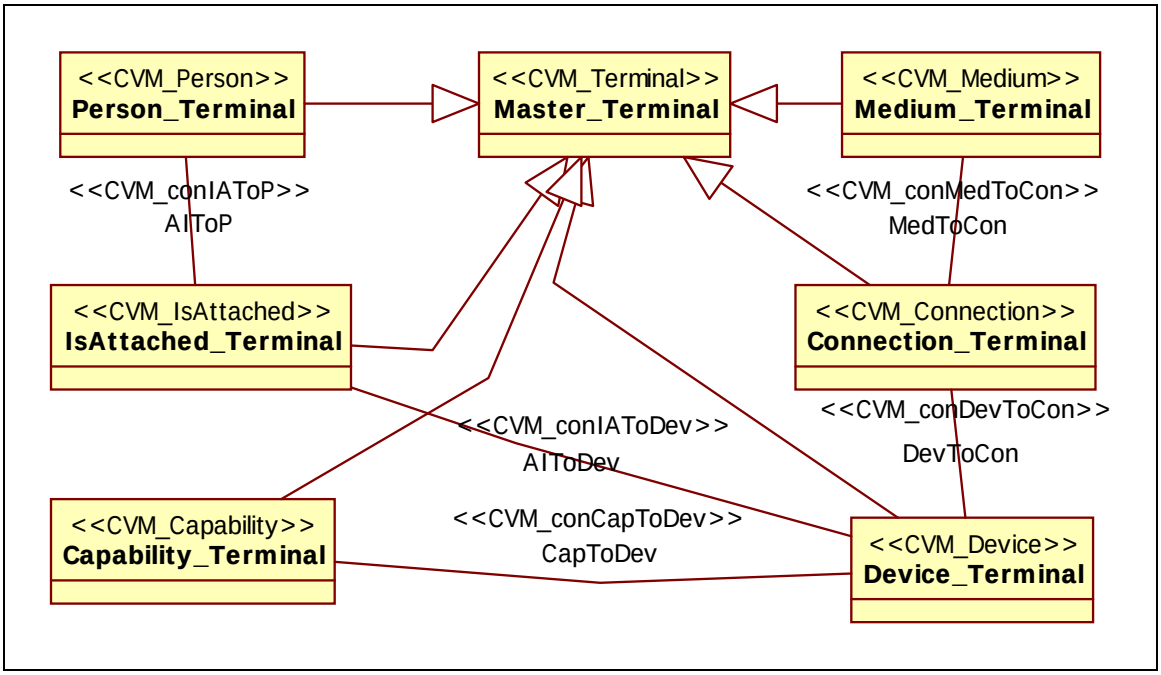


Fig C.1 Minimal Class Diagram Of The Modeling Construction (Modeling Constructing Profile)

Description The classes in this diagram represent all the non-terminal and terminal symbols of the CML grammar, which is used in the creation of communication models in RRCommSSys. This is a version compliant to the UML profile for the purpose of a better understanding of GMF during the design phase.

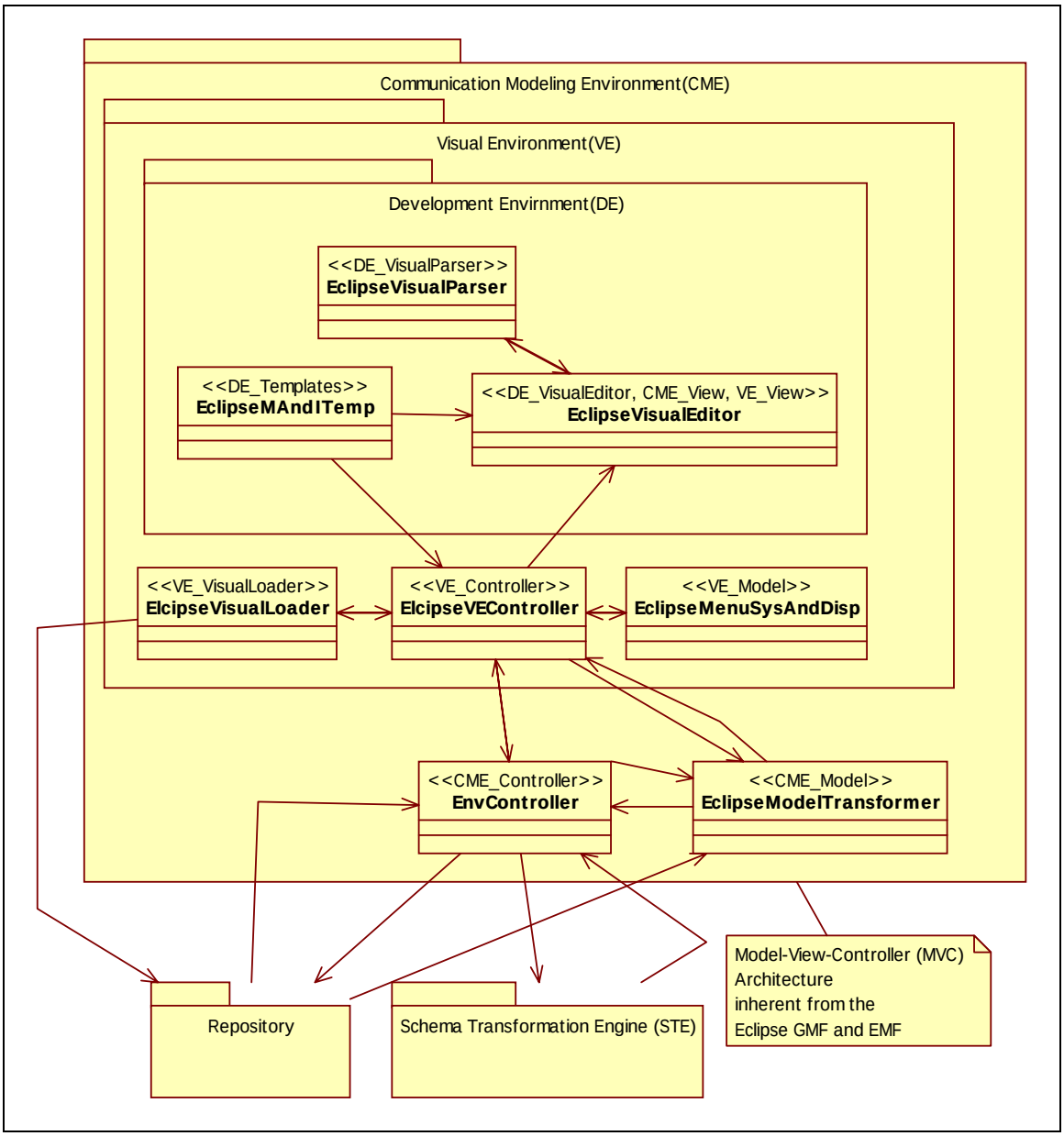


Fig C.3 Communication Modeling Environment

Description: The CME subsystem contains all the classes that are required for the modeling environment. This diagram show all the classes as well as all the all the dependencies. It also display the Model-View-Controller architecture pattern. This pattern is inherited from Eclipse.

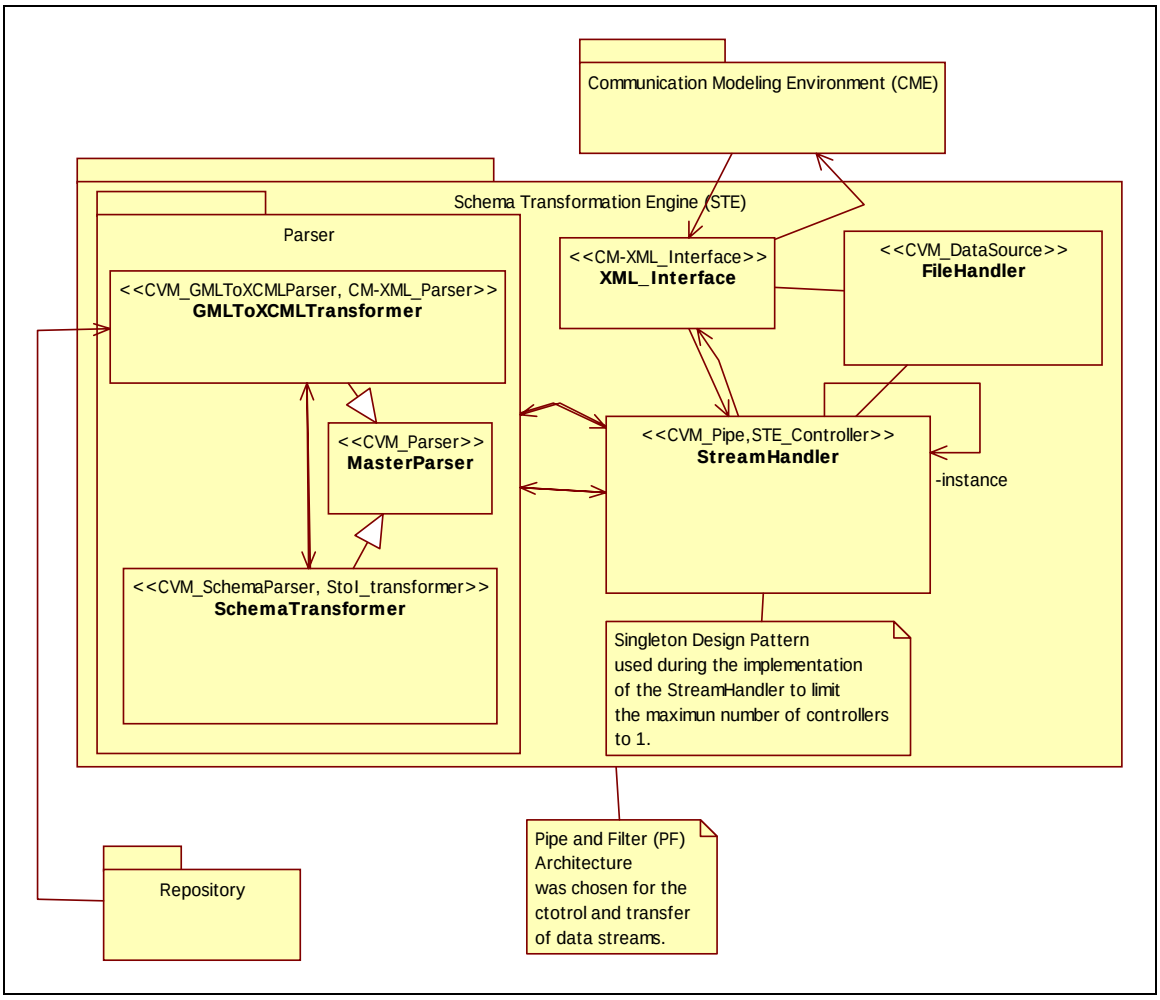


Fig C.4 Class Diagram For The Schema Transformation Engine

Description: This is the class diagram for the Schema transformation Engine. Here are all the classes that will be implemented for the transformation of schemas into instances. And architecture pattern of Pipe and Filter is used for controlling of the stream of data between parsers. Also a Singleton design pattern is used to limit the instances of our controller to one.

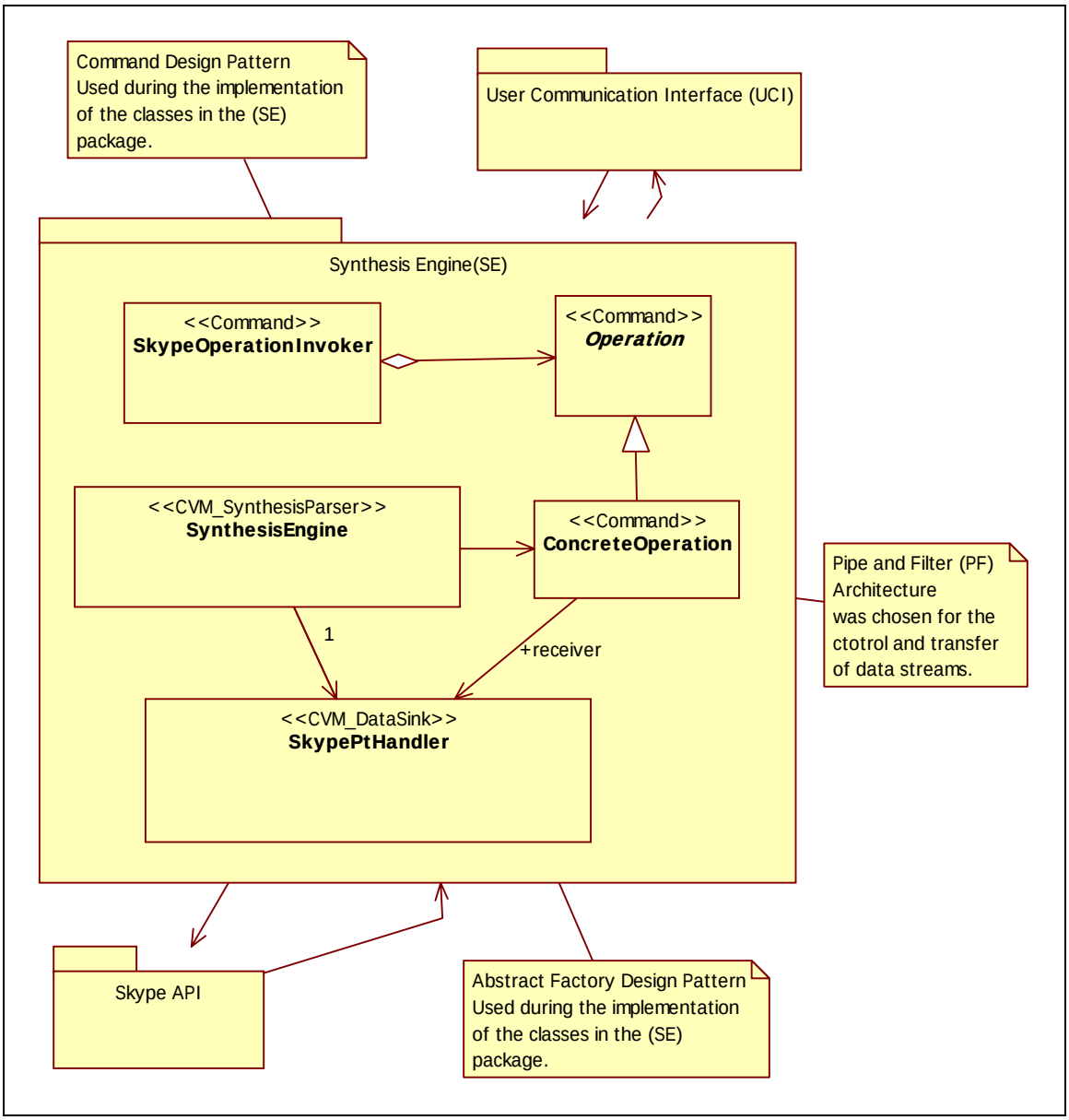


Fig C.5 Package Diagram Synthesis Engine Subsystem.

Description: This is the class diagram for the Synthesis Engine subsystem. This are the classes that will be implemented for the of the execution of the XCML. The Command design pattern is used for more control over the calls that we will be making. Also the Abstract Factory design pattern is used to allow in future development the replacement of the Skype API for another platform, as well as to allow the system to run on multiple operating systems.

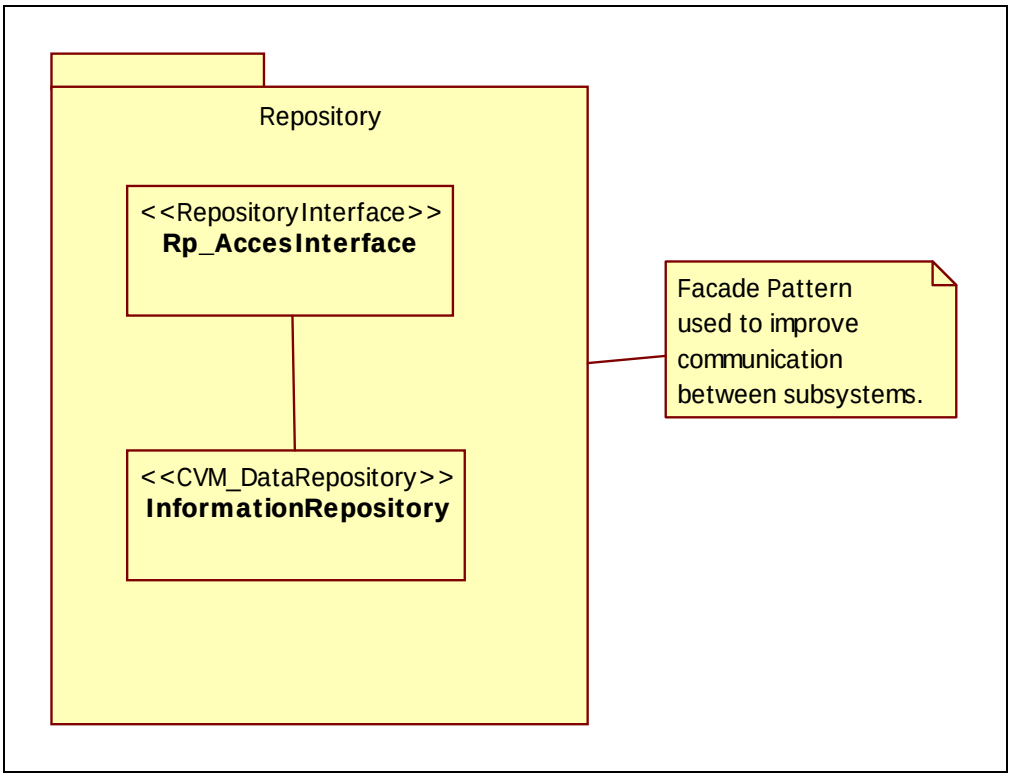


Fig.F.6 Repository Package Class Diagram

Description: This is the class diagram for the repository that will be implemented on our project. This repository will hold the metadata as well as the information for parsing our files.

8.4 Appendix D – Sequence Diagrams

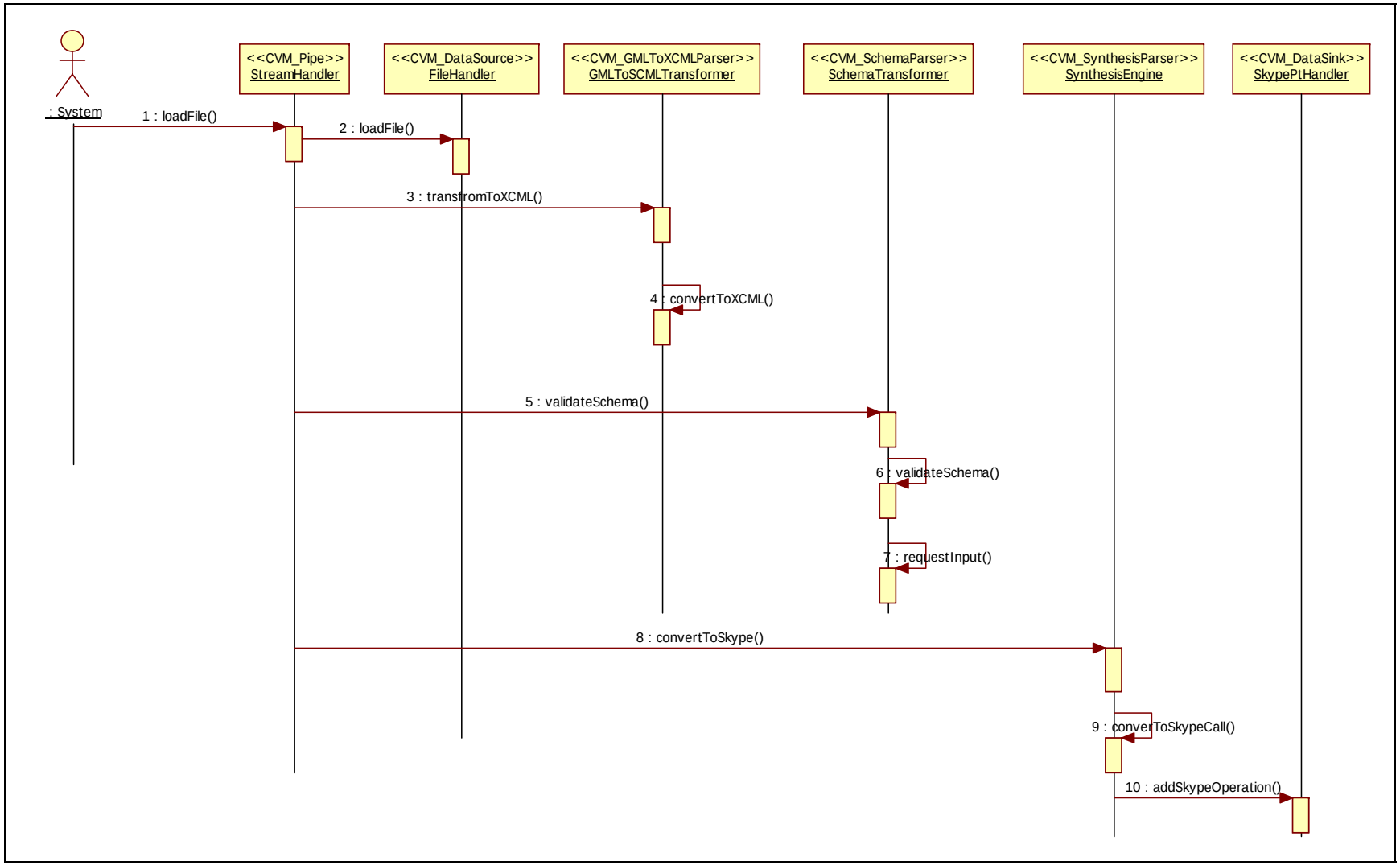


Fig D.3 Sequence For a Schema Transformation

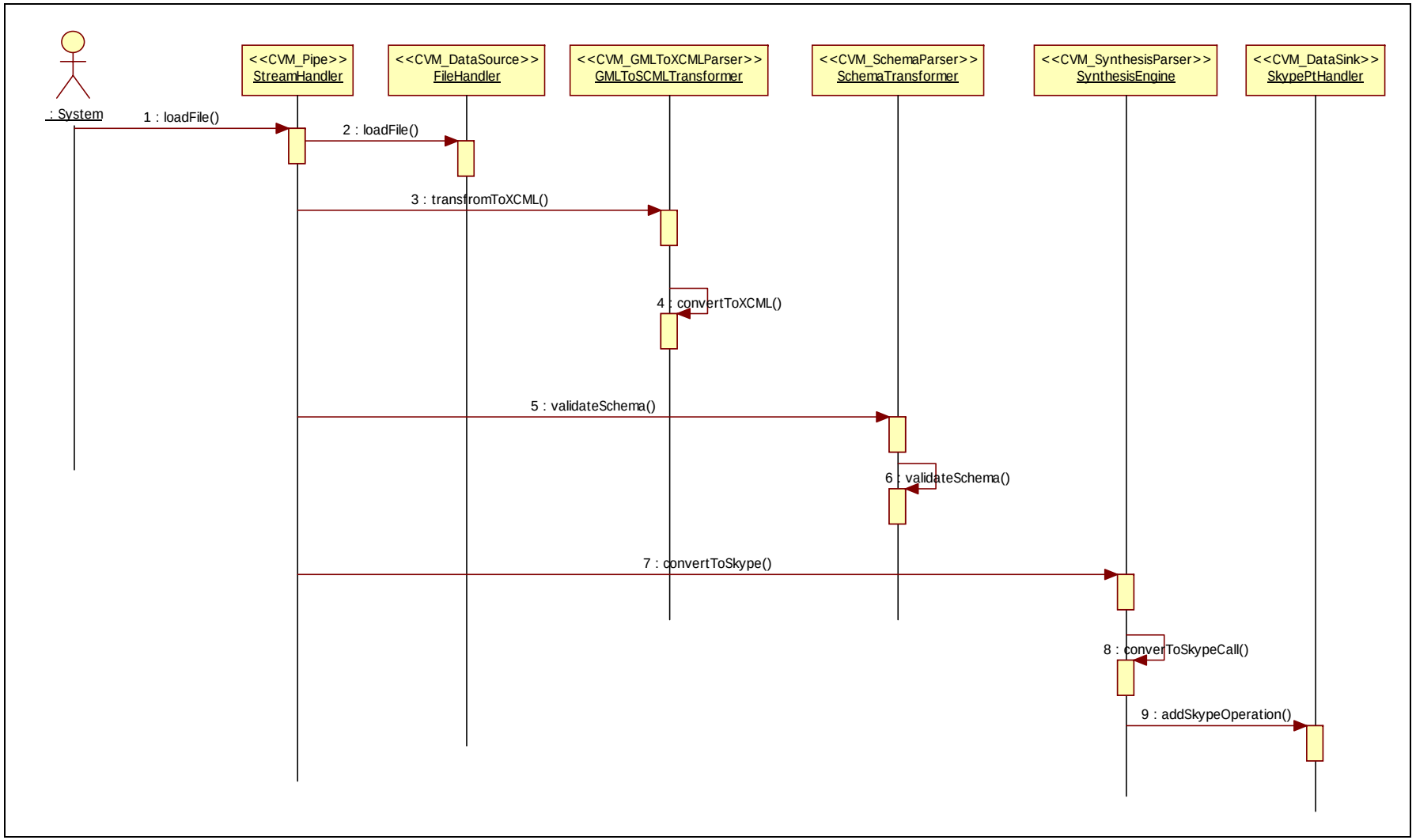


Fig. D.4 Instance Execution

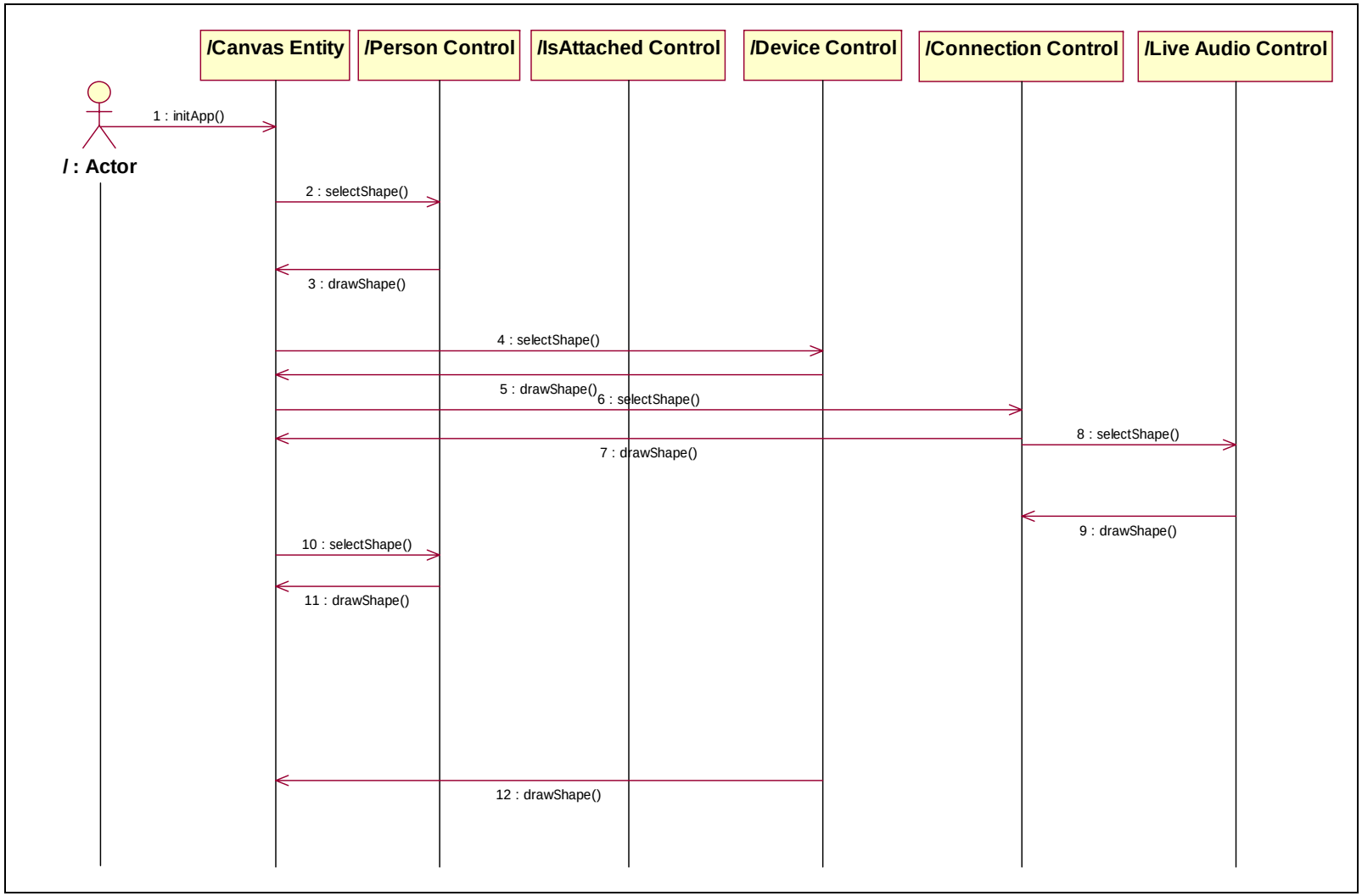


Fig D.5 Sequence Diagram For Voice Call

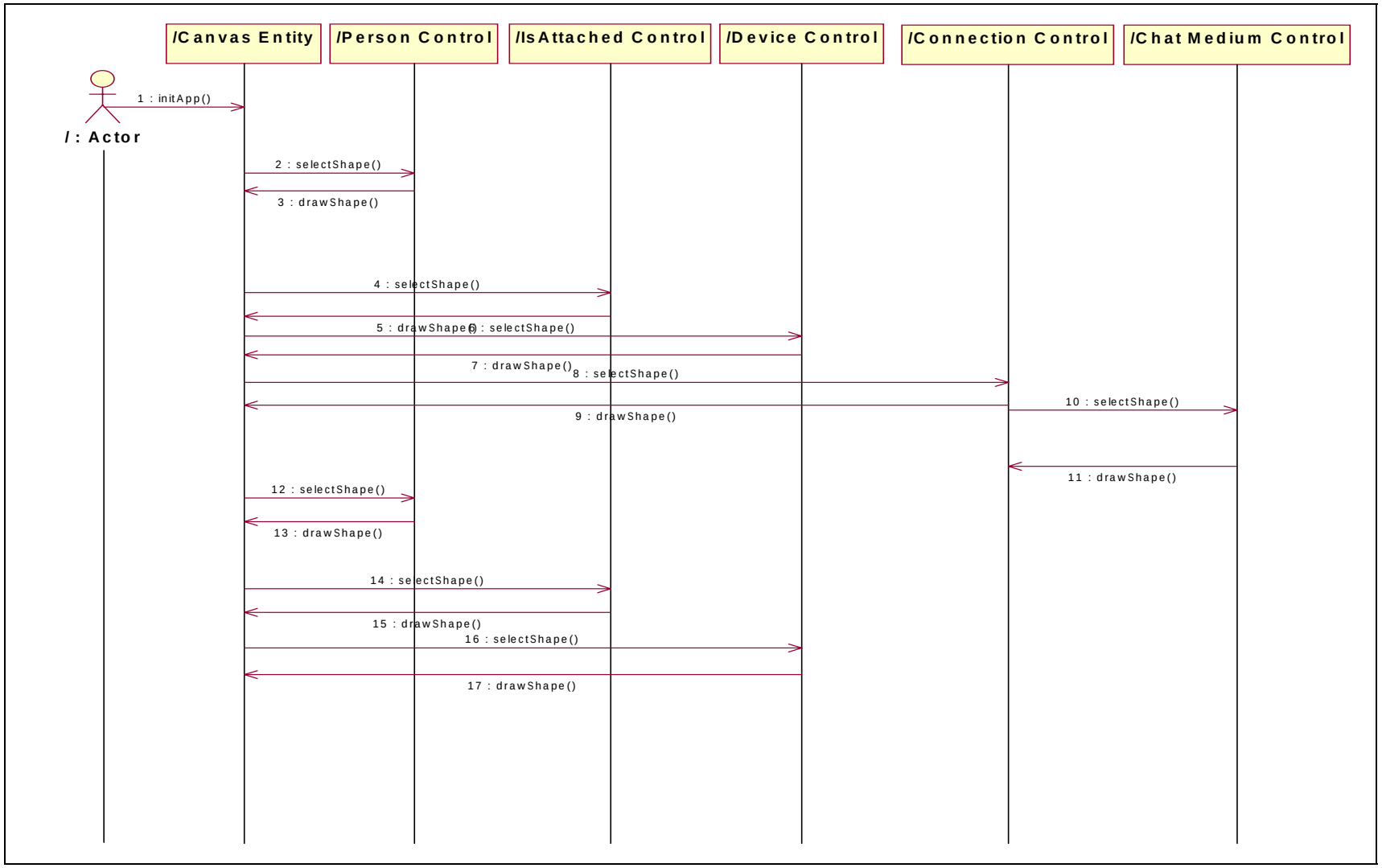


Fig D.6 Sequence For Chat Message.

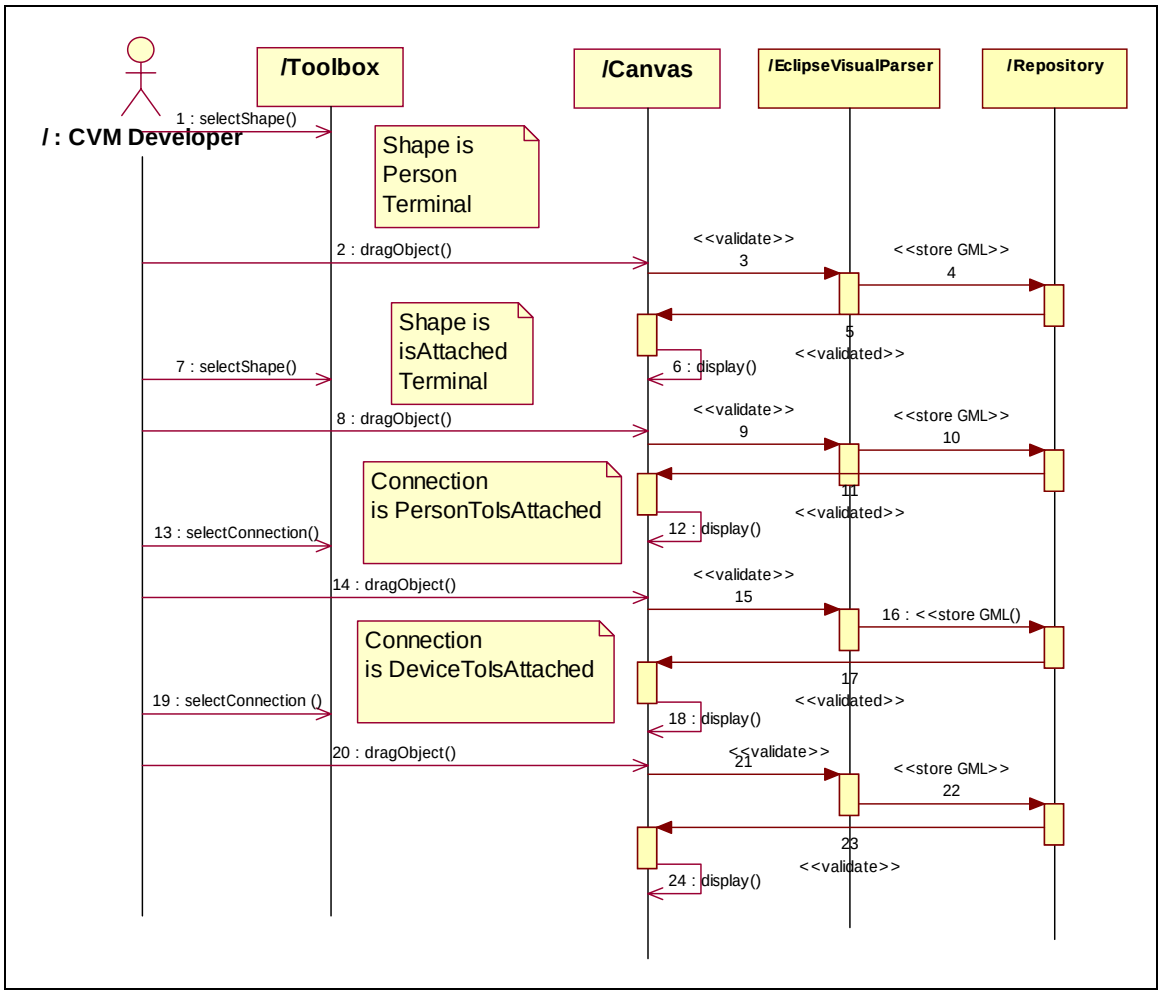


Fig D.7 Sequence For Create Local Non Terminal

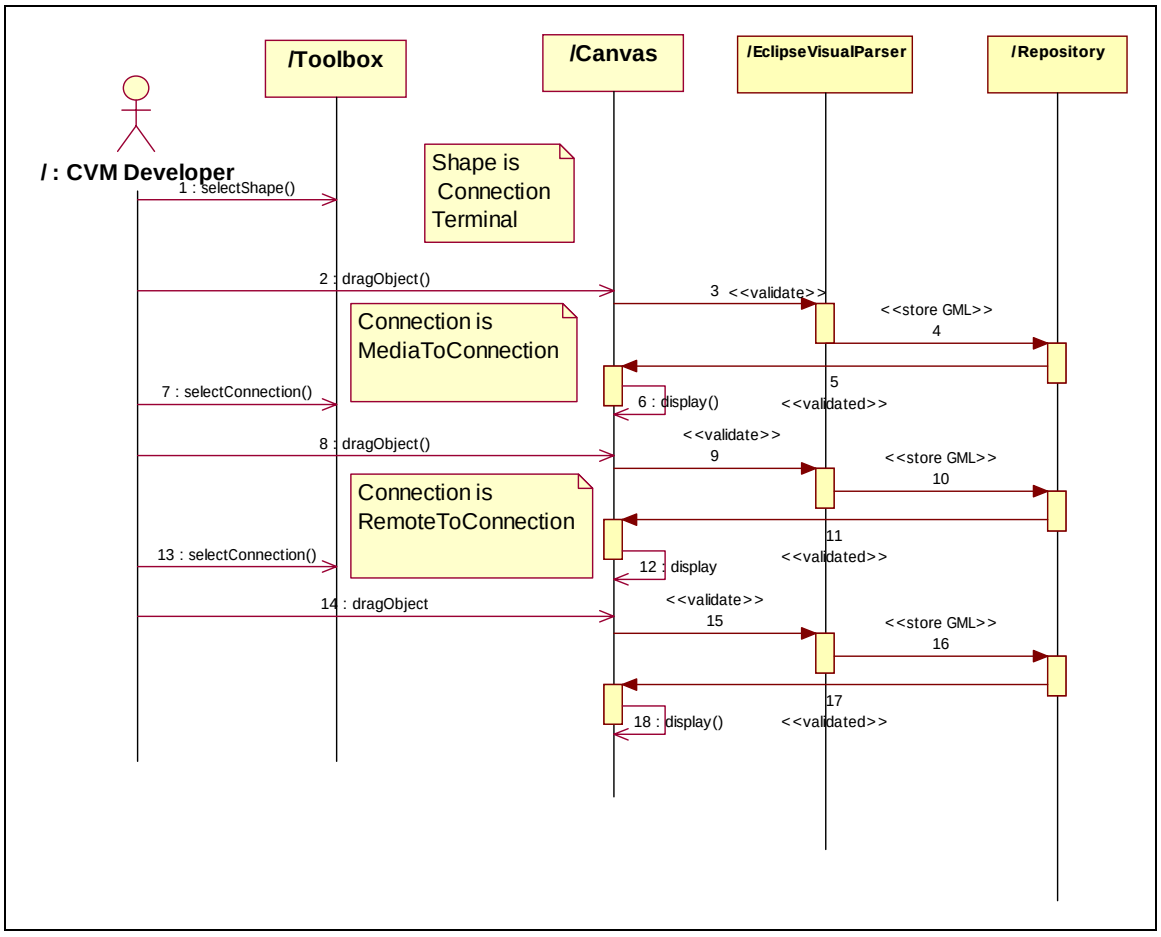


Fig D.8 Sequence For Create Connection

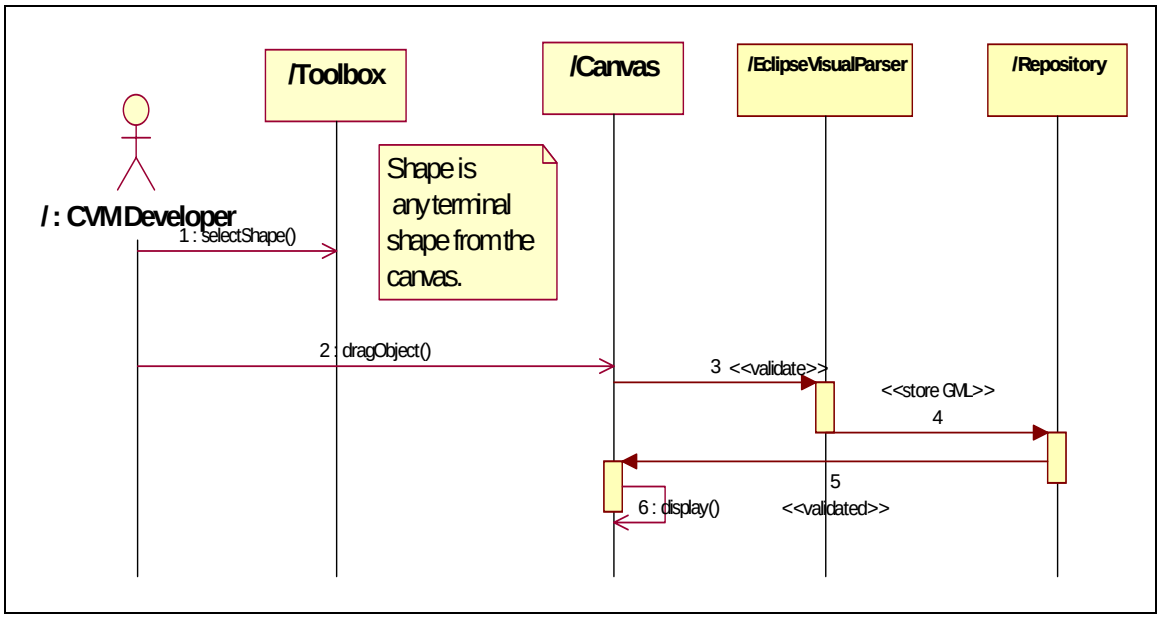


Fig D.9 Sequence For Create Terminal.

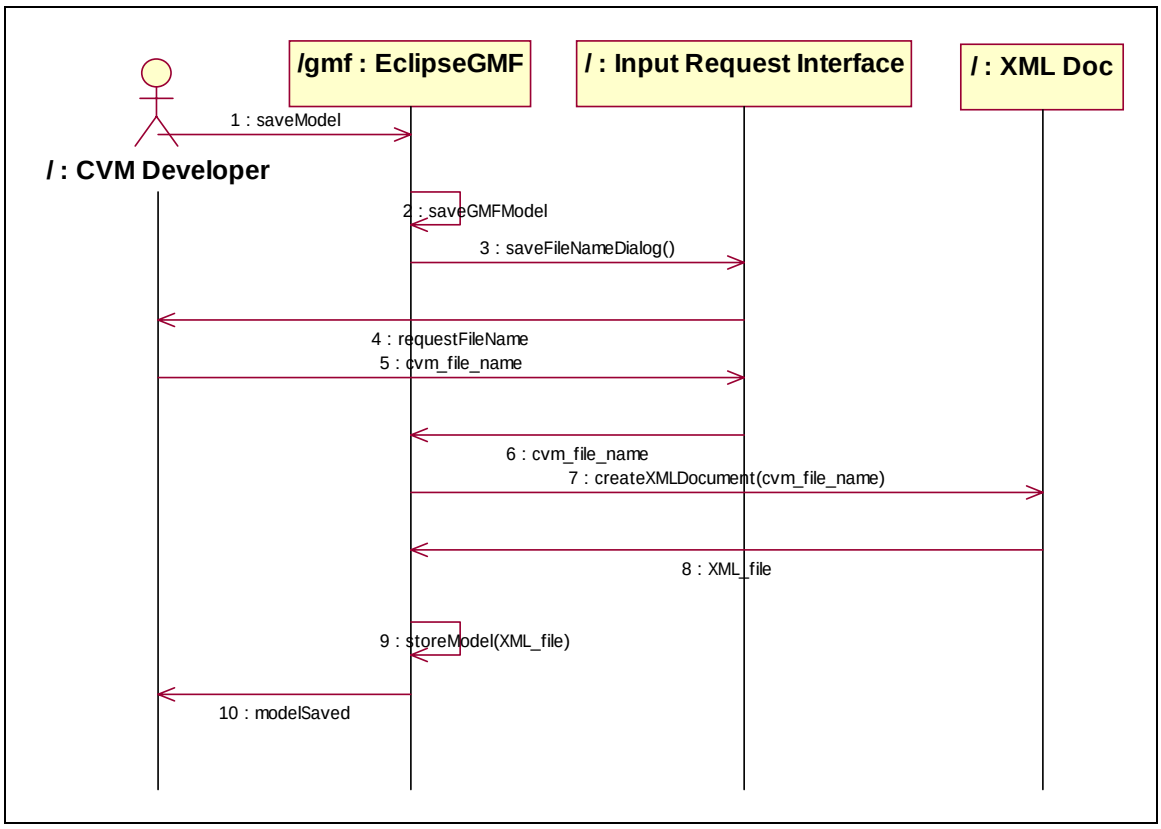


Fig D.10 Sequence For Save File GMF

8.5 Appendix E – User Interfaces

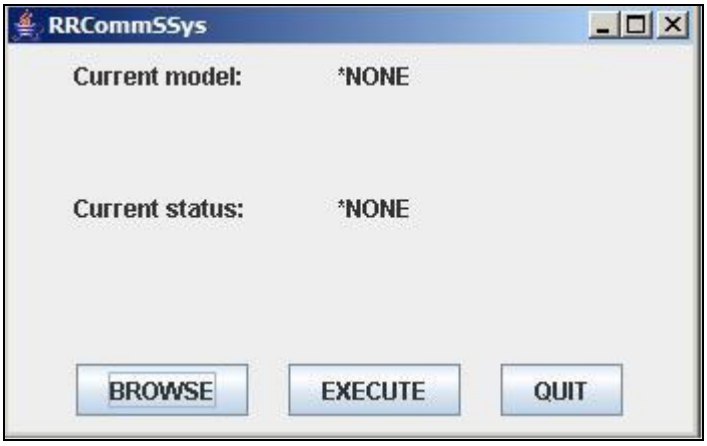


Fig. E1.1 Load Request Form.

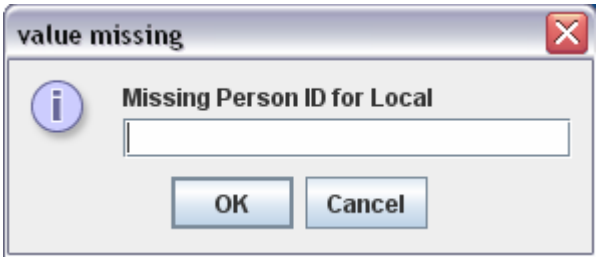


Fig. E1.2 Impute Request Form.

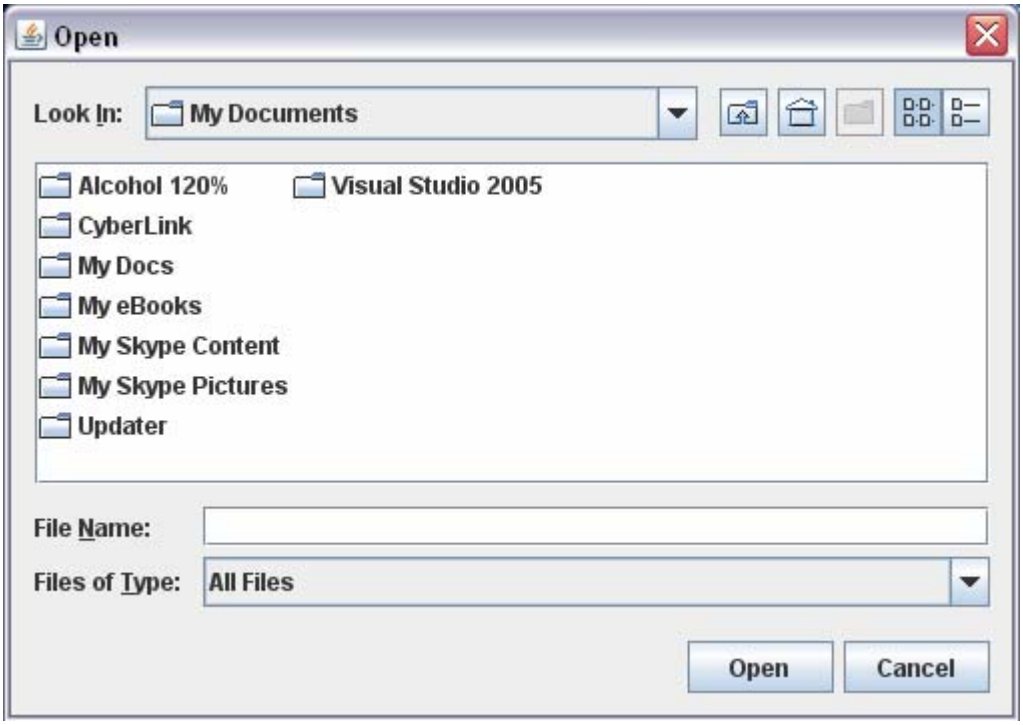


Fig. E1.3 File System open file Dialog.

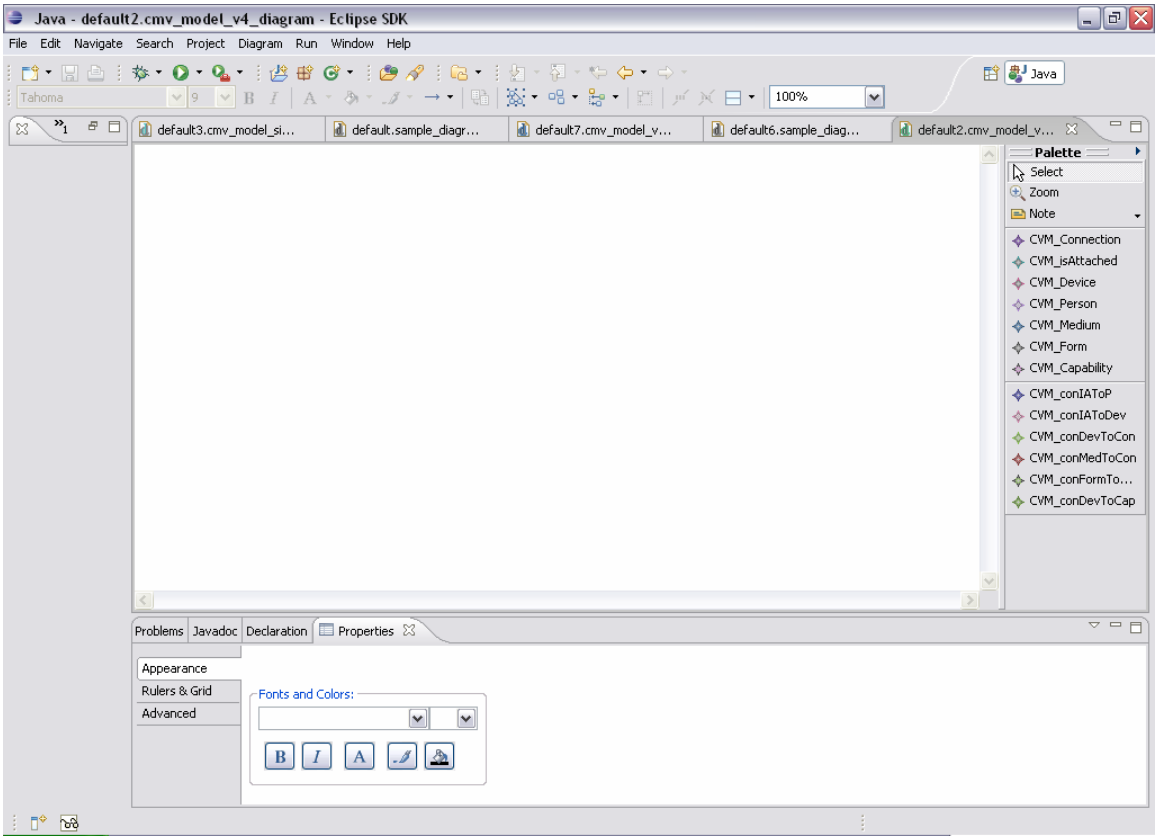


Fig. E1.4 Visual Modeling Environment

8.6 Appendix F – Detailed Class Diagram

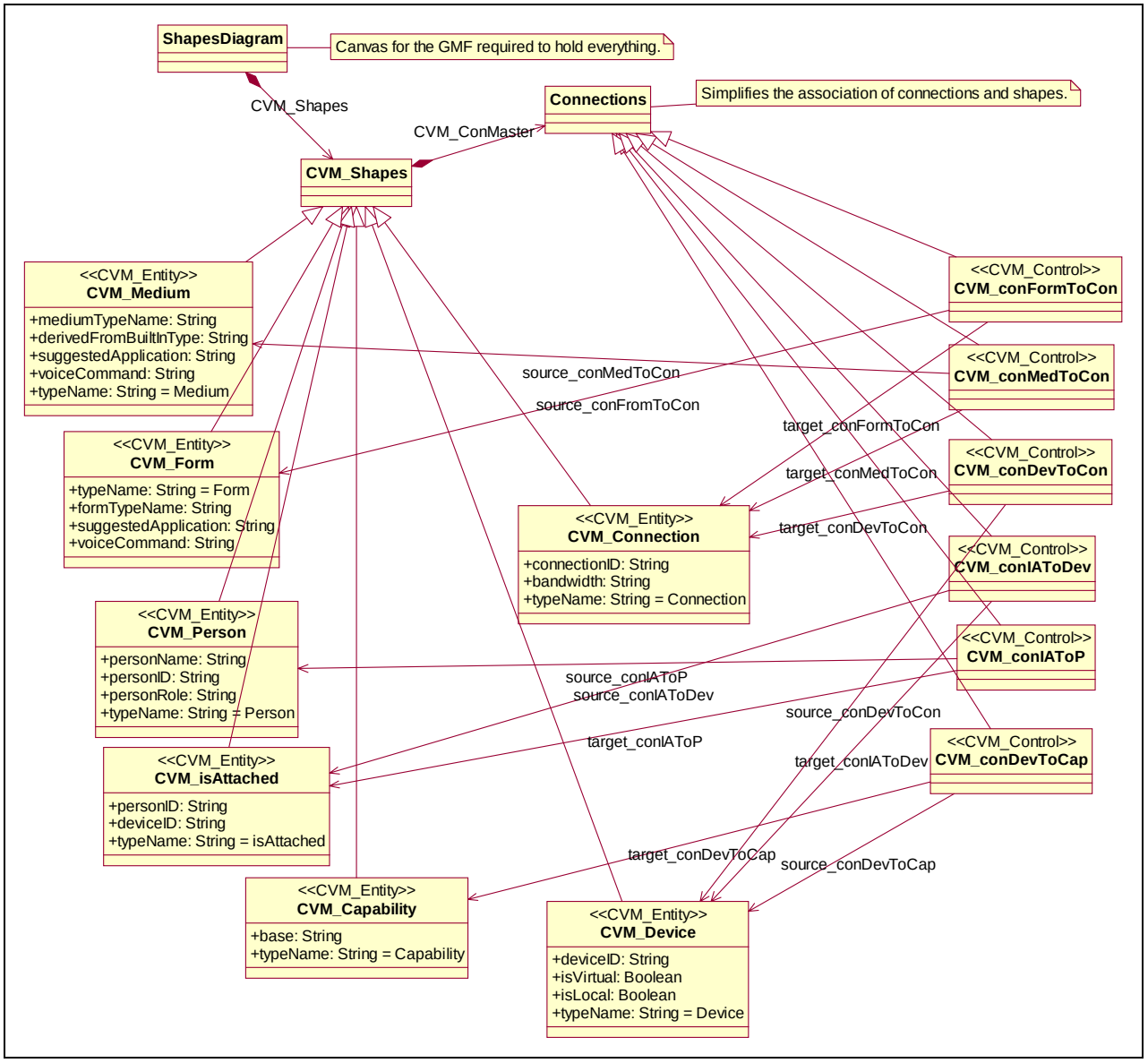


Fig. F.1 Detailed class diagram of the Model Creation Subsystem As Used By GMF.

Description The classes in this diagram represent all the non-terminal and terminal symbols of the CML grammar, which is used in the creation of communication models in RRCommSSys.

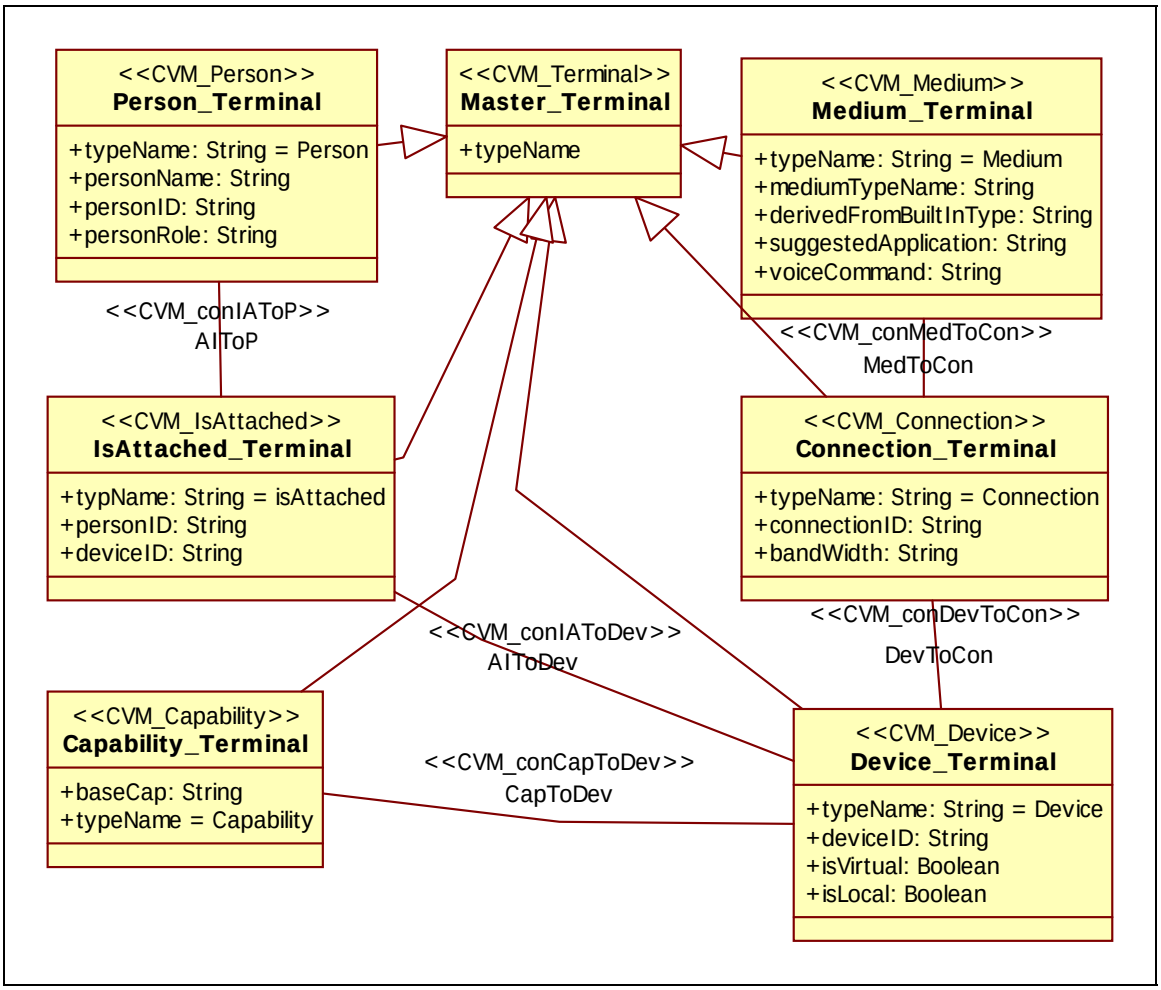


Fig F.2 Detailed class diagram of the Model Creation Subsystem From Profile

Description The classes in this diagram represent all the non-terminal and terminal symbols of the CML grammar, which is used in the creation of communication models in RRCommSSys. This is a version compliant to the UML profile for the purpose of a better understanding of GMF during the design phase.

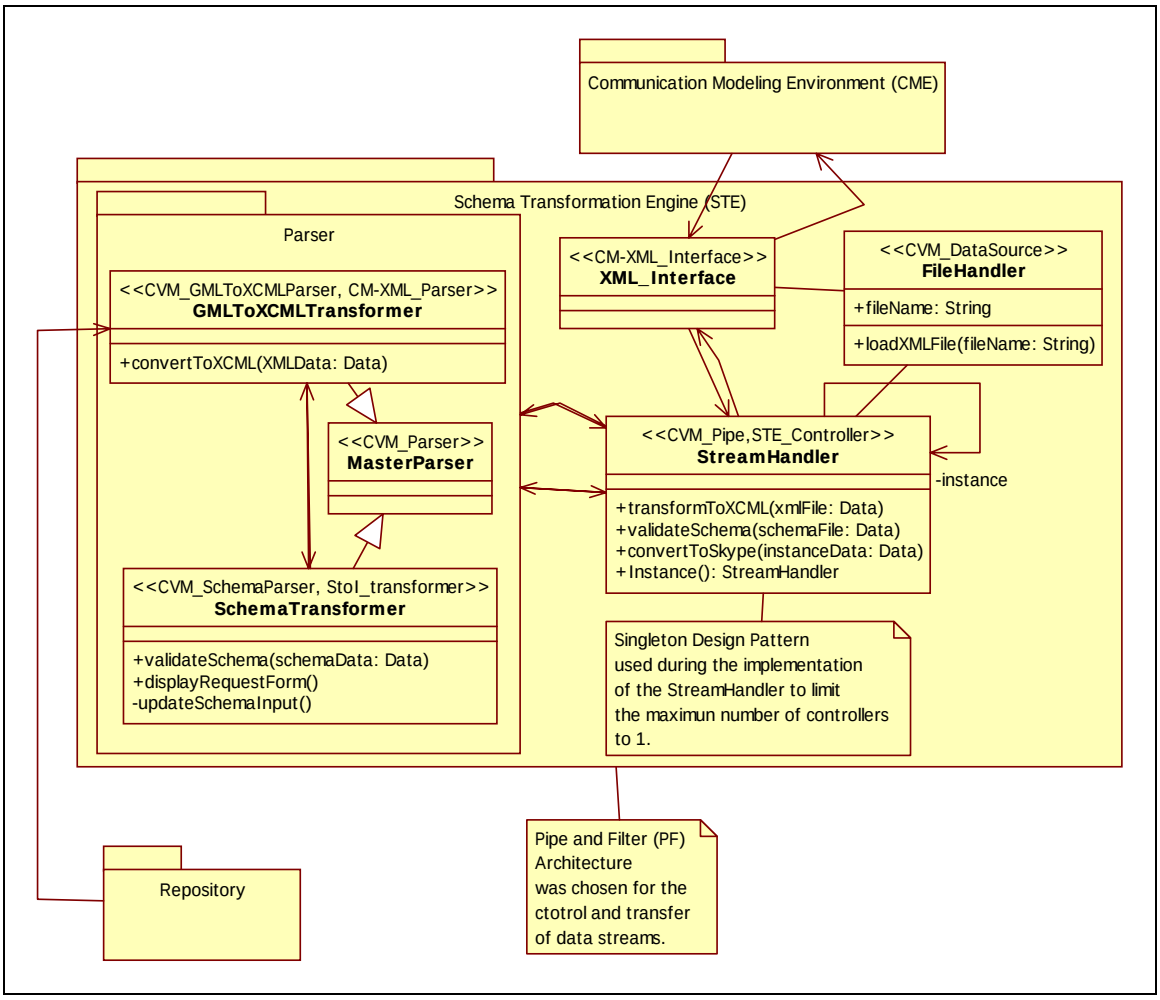


Fig F.4 Class Diagram For The Schema Transformation Engine

Description: This is the class diagram for the Schema transformation Engine. Here are all the classes that will be implemented for the transformation of schemas into instances. And architecture pattern of Pipe and Filter is used for controlling of the stream of data between parsers. Also a Singleton design pattern is used to limit the instances of our controller to one.

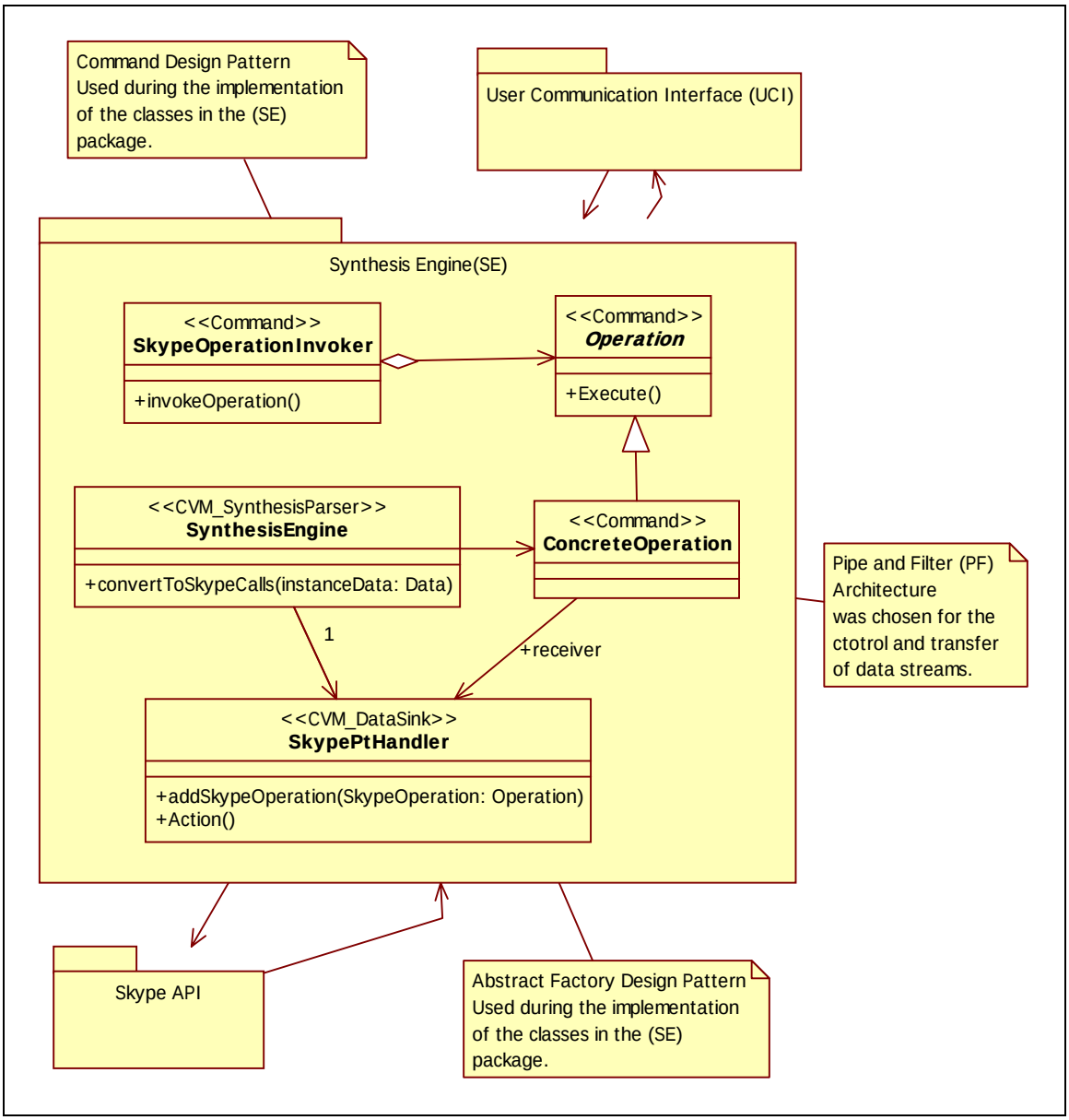


Fig F.5 Package Diagram Synthesis Engine Subsystem.

Description: This is the class diagram for the Synthesis Engine subsystem. This are the classes that will be implemented for the of the execution of the XCML. The Command design pattern is used for more control over the calls that we will be making. Also the Abstract Factory design pattern is used to allow in future development the replacement of the Skype API for another platform, as well as to allow the system to run on multiple operating systems.

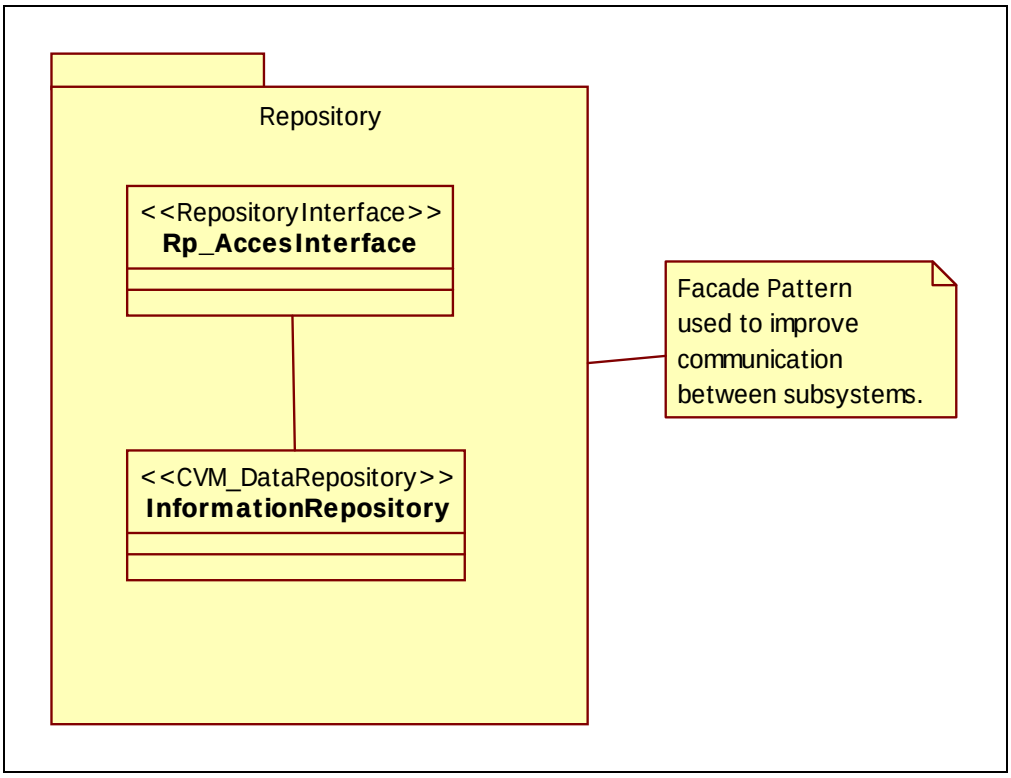


Fig.F.6 Repository Package Class Diagram

Description: This is the class diagram for the repository that will be implemented on our project. This repository will hold the metadata as well as the information for parsing our files.

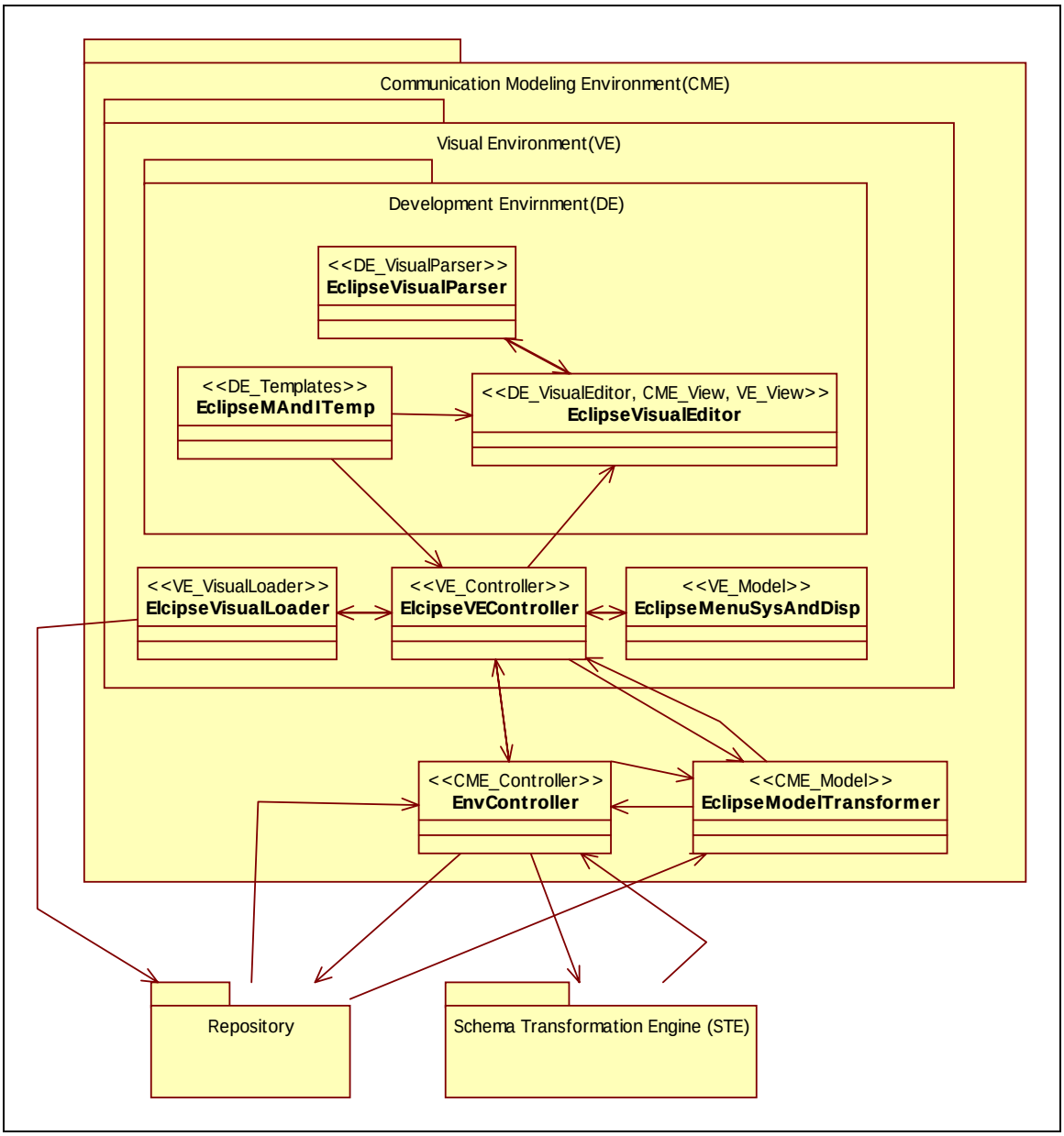


Fig F.7 Communication Modeling Environment

Description: The CME subsystem contains all the classes that are required for the modeling environment. This diagram show all the classes as well as all the all the dependencies. It also display the Model-View-Controller architecture pattern. This pattern is inherited from Eclipse.

8.7 Appendix G – Class Interfaces

```
package execution.datasink;

/**

* Command call specific to a platform

* @author amortiz

*

*/

public class ConcreteOperation extends CommandCall {

    public void Execute();

}
```

```
package execution.datasink;

/**

* Class providing abstract factory pattern. Given the platform

* it returns the proper object for that platform

* @author amortiz

*

*/

public abstract class CommandCall {

    public abstract void Execute();

    public static final CommandCall getCommand(int type, int platform);

}
```

```
/**
```

```
* Encapsulates the input and output of the filter processes
* @author amortiz
*
*/
package execution.datasource;
public class FileHandler {
    private String fileName;
    public void loadXMLFile(String fileName) ;
}
```

```
package execution.filter;
/**
* Parser that transforms GML to XCML model
* @author amortiz
*
*/

public class GMLToXCMLTransformer extends MasterParser {
    public void convertToXCML(Data XMLData);
}
```

```
package execution.filter;
/**
* All parser extend this class, representing the input
* processing, and output of a model
* @author amortiz
```

```
*
*/
public class MasterParser {
    protected Data input;
    protected Data output;
}
```

```
package execution.filter;
/**
* All parser extend this class, representing the input
* processing, and output of a model
* @author amortiz
*
*/
public class SchemaTransformer extends MasterParser {
    public void validateSchema(Data schemaData;

    public void displayRequestForm();

    private void updateSchemaInput();
}
```

```
package execution.filter;
/**
* It creats a series of calls to the underlying platform based
```

```
* on an input instance

* @author amortiz

*

*/

public class SynthesisEngine extends MasterParser {

    public void convertToSkypeCalls(Data instanceData);

}
```

```
package execution.pipe;

/**

*

* It is the entry point of the application. It gets the GCML file from
the user

* and passes it along to all the required filters to finally deliver it
to the

* data sink

* @author amortiz

*

*/

public class StreamHandler {

    private StreamHandler instance;

    public void transformToXCML(Data xmlFile);


    public void validateSchema(Data schemaFile);


    public void convertToSkype(Data instanceData);
```

```
    public StreamHandler Instance() ;
}
```

```
package modeling.model;
import java.util.ArrayList;
/**
* represents a device capability on the communication model
* @author amortiz
*
*/
public class Capability_Terminal extends Master_Terminal {
    public String baseCap;
    public final static String type = "Capability";
    public Capability_Terminal(String baseCap);
   //mutators and accessors
    public String getBaseCap() ;
    public void setBaseCap(String baseCap);


}
```

```
package modeling.model;
import java.util.ArrayList;
/**
* represents an Connection on the communication model
```

```
* @author amortiz
*
*/
public class Connection_Terminal extends Master_Terminal {
    private String connectionID;
    private String bandWidth;
    public final static String type = "Connection";
    public Connection_Terminal(String connectionID, String bandWidth);
   //mutators and accessors
    public String getBandWidth();
    public void setBandWidth(String bandWidth);
    public String getConnectionID();
    public void setConnectionID(String connectionID);
}
```

```
package modeling.model;
import java.util.ArrayList;


/**
* represents a Device on the communication model
* @author amortiz
*
*/
public class Device_Terminal extends Master_Terminal {
    public final static String type = "Device";
```

```
    private String deviceID;

    private Boolean isVirtual;

    private Boolean isLocal;

    public Device_Terminal(String deviceID, Boolean isVirtual, Boolean
isLocal);

   //mutators and accessors

    public String getDeviceID();

    public void setDeviceID(String deviceID);

    public Boolean getIsLocal();

    public void setIsLocal(Boolean isLocal);

    public Boolean getIsVirtual();

    public void setIsVirtual(Boolean isVirtual);


}
```

```
package modeling.model;

import java.util.ArrayList;

/**

* represents an IsAttached on the communication model

* @author amortiz

*

*/

public class IsAttached_Terminal extends Master_Terminal {

    public final static String type = "isAttached";

    private String personID;

    private String deviceID;
```

```
    public IsAttached_Terminal(String personID, String deviceID;
   //mutators and accessors
    public String getDeviceID();
    public void setDeviceID(String deviceID);
    public String getPersonID() ;
    public void setPersonID(String personID) ;

}
```

```
package modeling.model;
import java.util.ArrayList;
/**
* Represents every Terminal node of a communication model
* @author amortiz
*
*/
public abstract class Master_Terminal {
    private String typeName;
    public Master_Terminal(String type) ;
   //mutators and accessors
    public String getTypeName() ;
    public void setTypeName(String typeName);
}
```

```
package modeling.model;
```

```
import java.util.ArrayList;

/**

* represents a medium on the communication model

* @author amortiz

*

*/

public class Medium_Terminal extends Master_Terminal {

    public final static String type = "Medium";

    private String mediumTypeName;

    private String derivedFromBuiltInType;

    private String suggestedApplication;

    private String voiceCommand;

    public Medium_Terminal(String mediumTypeName, String
derivedFromBuiltInType, String suggestedApplication, String
voiceCommand);



   //mutators and accessors

    public String getDerivedFromBuiltInType();

    public void setDerivedFromBuiltInType(String
derivedFromBuiltInType);

    public String getMediumTypeName();

    public void setMediumTypeName(String mediumTypeName);

    public String getSuggestedApplication();

    public void setSuggestedApplication(String suggestedApplication);

    public String getVoiceCommand();

    public void setVoiceCommand(String voiceCommand);
```

```
}
```

```
package modeling.model;

import java.util.ArrayList;

/**

* represents a Person on the communication model

* @author amortiz

*

*/

public class Person_Terminal extends Master_Terminal {

    public final static String type = "Person";

    private String personName;

    private String personID;

    private String personRole;


    public Person_Terminal(String personName, String personID, String
personRole);

   //mutators and accessors

    public String getPersonID();

    public void setPersonID(String personID);

    public String getPersonName();

    public void setPersonName(String personName);

    public String getPersonRole() ;

    public void setPersonRole(String personRole);

}
```

8.8 Appendix H – Project Schedule

ID	Task Name	Start	Finish	Duration
1	Draw Gantt Chart	1/9/2007	1/9/2007	1d
2	Write Use Cases	1/9/2007	1/23/2007	11d
3	Validate Use Cases	1/24/2007	1/26/2007	3d
4	Identify Requirements	1/29/2007	2/6/2007	7d
5	Create Use case Model	1/29/2007	2/6/2007	7d
6	Create Object and Dynamic Models	2/7/2007	2/20/2007	10d
7	Create user interface mockups	1/29/2007	2/6/2007	7d
8	Write Scenarios	1/29/2007	2/6/2007	7d
9	Milestone 1	2/21/2007	2/21/2007	0d
10	Subsystem Decomposition	2/21/2007	2/26/2007	4d
11	Create Modeling environment	2/21/2007	2/26/2007	4d
12	Write Parsers and tranformers	3/2/2007	3/12/2007	7d
13	Validate transformations	3/13/2007	3/21/2007	7d
14	Generate code based on Models	2/27/2007	3/1/2007	3d
15	Create calls to Skype interface	3/22/2007	3/22/2007	1d
16	Milestone 2	3/27/2007	3/27/2007	0d
17	System Test	3/27/2007	4/10/2007	11d
18	Subsystem Tests	3/27/2007	4/10/2007	11d
19	Evaluation of tests	4/11/2007	4/18/2007	6d
20	Milestone 3	4/19/2007	4/19/2007	0d

8.9 Appendix I – Diary of Meetings

Place: Classroom
Date: 1/23 **Facilitator:** Alejandro
Start: 9:00 PM **Attending:** Alejandro, Frank, Ariel
End: 9:30 PM **Minute Taker:** Ariel

1. **Objective**

   Project kick off meeting.

2. **Status**

   Everyone participated in coming up with ideas for the project.

3. **Discussion**

   Understanding of the problem statement. Anticipated facing issues with the Eclipse GMF solution due to its yet immature state.

Place: ECS 212
Date: 1/24 **Facilitator:** Alejandro
Start: 4:00 PM **Attending:** Alejandro, Frank, Ariel
End: 4:30 PM **Minute Taker:** Ariel

1. **Objective**

   Meet TA Tuan Cameron to have a hands-on session on GMF.

2. **Status**

   Asked specific questions to Tuan for the creation of the communication graphical environment.

3. **Discussion**

   Saw a live demonstration of GMF usage. Answered questions related to GMF links. Realized GMF non-resiliency to changes in the class diagram.

4. **Tasks**

   All team members will propose a class diagram for the graphical environment.

Place: Undergraduate Lab
Date: 1/29 **Facilitator:** Alejandro
Start: 9:00 PM **Attending:** Alejandro, Ariel
End: 9:30 PM **Minute Taker:** Ariel

1. **Objective**

   Revision of the graphical model class diagram. First GMF hands-on.

2. **Status**

   Discussed the class diagram and made improvements. Completed a cycle in Eclipse using GMF for implementing the graphical environment.

3. **Discussion**

   Errors while running the GMF model. After running, issues when linking shapes in GMF. Open issues to be solved later with the TA.

Place: Undergraduate Lab
Date: 2/2 **Facilitator:** Alejandro
Start: 5:00 PM **Attending:** Alejandro, Frank, Ariel
End: 8:30 PM **Minute Taker:** Ariel

1. **Objective**

   Focus on deliverable 1 due on 2/20.

2. **Status**

   Wrote down a list of use cases for the graphical and execution environments, and split the detailed description of each one to be done by all team members.

3. **Tasks**

   Alejandro – All uses cases for the terminal shapes; Frank – uses cases for model execution; Ariel – generic terminal shape and persistency uses cases.

Place: Undergraduate Lab
Date: 2/13 **Facilitator:** Alejandro
Start: 5:30 PM **Attending:** Alejandro, Frank, Ariel
End: 6:10 PM **Minute Taker:** Ariel

1. **Objective**

   Slide preparation for after-Exam 1 presentation.

2. **Status**

   Compiled one uses case and its sequence diagram to be presented in class.

3. **Discussion**

   A portion of the 4-slide presentation was allotted for presenting by each team member.

Place: Undergraduate Lab
Date: 2/16 **Facilitator:** Alejandro
Start: 2:00 PM **Attending:** Alejandro, Frank, Ariel
End: 5:30 PM **Minute Taker:** Ariel

1. **Objective**

   Focus on deliverable 1 due on 2/20.

2. **Status**

   Created a class diagram for the execution environment. Split up document sections for writing by the team members.

3. **Discussion**

   Each member will: 1) draw sequence diagrams, and depict scenarios for allotted use cases, 2) write various sections of the deliverable 1. All written material will be later combined in a single document for submission.

Place: Undergraduate Lab
Date: 3/16 **Facilitator:** Alejandro
Start: 5:00 PM **Attending:** Alejandro, Frank, Ariel
End: 7:00 PM **Minute Taker:** Ariel

1. **Objective**

   Focus on deliverable 2 due on 3/27.

2. **Status**

   Brain storm session for deciding the architectural and design patterns to be used in the system design. Created a list of high-level packages that are going to be part of the design. Split up document sections for writing by each team members.

3. **Discussion**

   The following patterns were proposed

   - Architectural: MVC, Pipe and Filter.
   - Design: Façade, Abstract factory, Command, others.

   Also, the following packages were considered:

   Packages: Filter, Data source, Pipe, Data sink, Model, View, Controller.

   The split up of the document sections are as follows:

   - Ariel: concentrates on chapter 1
   - Alejandro: concentrates on chapter 2
   - Frank: concentrates on chapter 3

   As well, we concentrated on fixing mistakes on deliverable 1, and focused on the new class diagram and profiles.

   Next meeting is going to be held on 3/20/2007 at 4:00PM, same place, for discussing the UML profiles of chapter 2.

Place: Undergraduate Lab
Date: 3/20 **Facilitator:** Alejandro
Start: 4:00 PM **Attending:** Alejandro, Frank, Ariel
End: 6:00 PM **Minute Taker:** Ariel

1. **Objective**

   Focus on UML profiles for deliverable 2.

2. **Status**

   Two UML profiles were proposed. One for the communication model creation environment, and the other one for the execution of the model. A brainstorm session was carried out for the model execution UML profile.

3. **Discussion**

   A UML profile was depicted containing the following meta-classes: Parser, Filter, Pipe, Data Sink, Data Source. This model is yet to be formalized by Frank.

Place: Undergraduate Lab
Date: 3/21 **Facilitator:** Alejandro
Start: 3:00 PM **Attending:** Alejandro, Frank, Ariel
End: 5:00 PM **Minute Taker:** Ariel

1. **Objective**

   Focus on UML profiles and security use case for deliverable 2.

2. **Status**

   Two UML profiles, model creation and execution, were depicted and drawn on paper. We discussed the correctness of the profiles and agreed on the meta-classes contained in each one. We also discussed about the security use case, the misuse case, which we also agree on one.

3. **Discussion**

   The UML profiles were formalized following the UML 2 notation.
   The security use case that is to be included in the document is related to a denial of service (DOS) scenario, in which a single Skype user is tried to be contacted a great number of times in a single communication model. Since that could congest the network, and may possibly result in a DOS situation, the system will prevent such scenario from happening in the first place. Similarly, a sub-case of this is to call ourselves in a single communication model. This is also not permitted.

Place: Undergraduate Lab
Date: 3/26 **Facilitator:** Alejandro
Start: 4:00 PM **Attending:** Alejandro, Frank, Ariel
End: 5:00 PM **Minute Taker:** Ariel

1. **Objective**

   Focus on finalizing software design document (deliverable 2).

2. **Status**

   Missing sections of the document were split up among the team members.

   Discussion

   The current status of the SDD document was reviewed. Missing sections were identified. Few sections were missing from the detailed design especially. We worked on the generative architecture of the system. The rest was left as homework.

Place: Undergraduate Lab
Date: 3/27 **Facilitator:** Alejandro
Start: 2:30 PM **Attending:** Alejandro, Frank, Ariel
End: 5:00 PM **Minute Taker:** Ariel

1. **Objective**

   Putting it all together in the design document (deliverable 2), and outline the presentation slides.

2. **Status**

   The SDD document was polished. As well, the presentation slides were finalized.

3. **Discussion**

   Printed out final version of deliverable 2 document for submission. Selected what we believed were the most important points of the document to be included in the presentation.

Place: Undergraduate Lab
Date: 3/30
Start: 5:00 PM
End: 6:30 PM

Facilitator: Alejandro
Attending: Alejandro, Frank, Ariel
Minute Taker: Ariel

1. **Objective**

   Focus on the final deliverable due on 4/13.

2. **Status**

   The document template provided by the professor was reviewed. The focus is on verifying, validating and implementing the system.

3. **Discussion**

   New sections were identified and discussed an outline as for the contents of those, especially in the implementation area. Also, the template may suffer changes as needed as advised by the professor.

Place: Undergraduate Lab
Date: 4/4
Start: 5:00 PM
End: 6:30 PM

Facilitator: Alejandro
Attending: Alejandro, Frank, Ariel
Minute Taker: Ariel

1. **Objective**

   More advances on the parsers' implementation were presented by Alejandro.

2. **Status**

   The G-CML file can be parsed, transformed to X-CML, validated, and instantiated. The next step is to make calls/send chat messages.

3. **Discussion**

   Sample parsed files were analyzed as for the validity of them against the X-CML schema. No apparent errors were found.

Place: Undergraduate Lab
Date: 4/6 **Facilitator:** Alejandro
Start: 5:00 PM **Attending:** Alejandro, Frank, Ariel
End: 6:30 PM **Minute Taker:** Ariel

1. **Objective**

   Implementational issues regarding the *Instant Messaging* part and Local/Remote person identification problem were discussed.

2. **Status**

   No explicit way to send an *Instant Message* seems to exist in the communication schema.

3. **Discussion**

   No optimal solution exist without modifying the communication schema. Potential workarounds are: 1) Similar to sending a file, specifying the file and the URL, and the file contains text that is read and fed in as an IM; certainly, not the most efficient solution. 2) Put the message in the mediumName attribute, leaving the URL blank, that is "borrowing" the attribute. We rather workaround 2. On the other hand, after some thought and clarifications from the Professor we have decided to use the Role of a person to say if that person is the Local or the remote user. Also have one for a meeting to prepare the presentation that will be done on the 4/17.

Place: Undergraduate Lab
Date: 4/9 **Facilitator:** Alejandro
Start: 4:00 PM **Attending:** Alejandro, Frank, Ariel
End: 5:30 PM **Minute Taker:** Ariel

1. **Objective**

   Review deliverable 3 document sections and user's guide.

2. **Status**

   The section writing is in progress. Expected end time is on 4/13.

3. **Discussion**

   Several sections for the implementation and test cases parts were reviewed. There is going to be one test case proposed per use case. We shall agree on the tabular format to record the test results in next meeting. Also, screenshots for the model execution part were collected to be included in the user's guide.

References

[1] Peter J. Clarke, et al. *A Declarative Approach for Specifying User-Centric Communication*. Collaborative Technologies and Systems, 2006.

[2] Peter J. Clarke. *Evolution of Software Design, Overview of MDSD*. Class notes, CEN 5064 Spring'07 Software Design class.

[3] http://en.wikipedia.org/wiki/Skype

[4] Eclipse Official Website