December 5, 2007
CEN 5011 – Advanced Software Engineering
Professor: Peter Clarke

User-Centric Communication Middleware

Team #1
Integrants:

1. **Batista, Raidel**
2. **Bhattacharya, Abhishek**
3. **Hernandez, Frank**
4. **Hernandez, Marylurdys**
5. **Monteiro, Eduardo**
6. **Zhao, Guangqiang**

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.

This document contains the requirements and system model documentations for a system based on the research done by FIU-SCIS, which we call "*User-Centric Communication Middleware*". This system will abstract the layer communication and doing so speed the development of future multimedia applications. The systems was implementing following the Unified Software Development approach.

Table of Contents

1. Introduction

In this chapter is the introduction to User-Centric Communication Middleware. 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 shall 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 follows a vertical development approach where each application is built on top of a low-level network abstraction. This development approach is what drives the cost of developing this application up. Also is this approach that slows down the pace of innovation of the new generation communication applications. The User-Centric Communication Middleware aims at solving this problem. The UCM aims to provide unified higher-level abstraction for the class of multimedia communication applications.

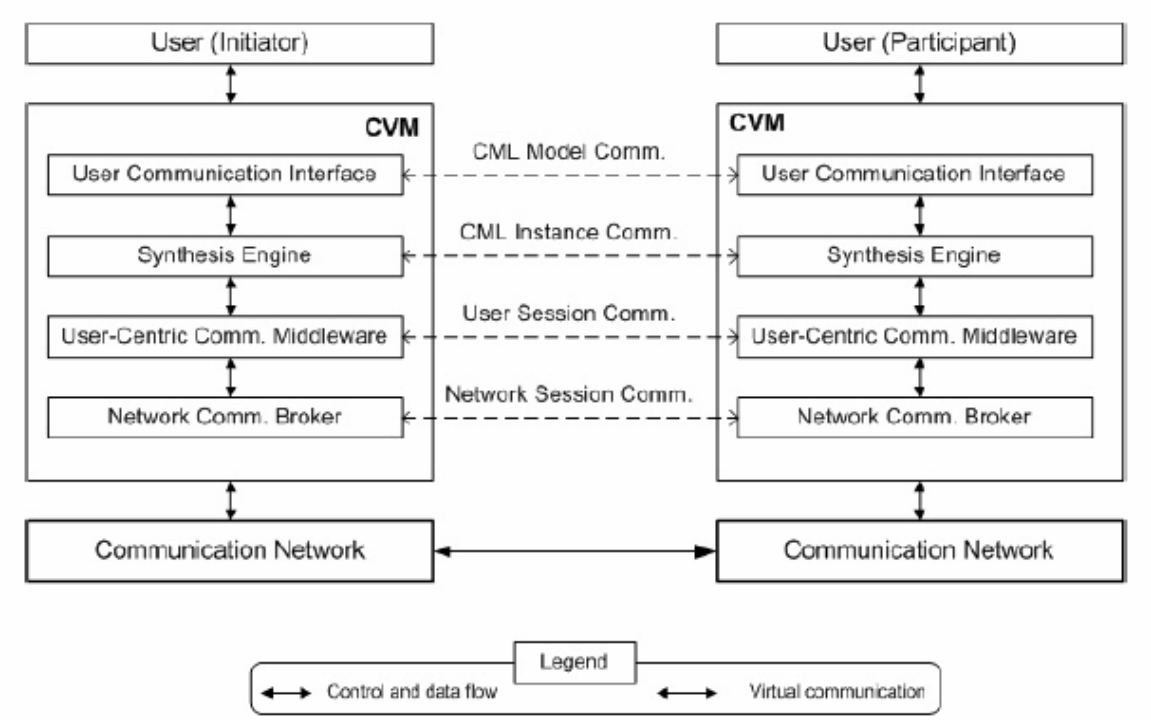


Fig. 1.1.1 CVM Design

Description: This is a representation of the CVM system. From this system we will be implementing the UCM layer. This layer will handle the communication between the Synthesis Engine and the Network Communication Broker.

1.2 Scope of System

The UCM system will practically allow a user to create communication applications without having to worry about creating the network protocol handlers every time. The system will lie under the Synthesis Engine, and over the Network Communication Broker. The system will then interpret the UCM control scripts output by the Synthesis Engine; it will log these control events, and then call NCB-specific commands. UCM will provide unified higher-level abstraction for the class of multimedia communication applications

1.3 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.

In addition, we eased the design of the system by using architectural and design patterns. The architectural patterns used are: repository and microkernel, whereas the design patterns: Command, Façade, Singleton, and Strategy.

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, and sequence diagrams, UML profiles. The following figure shows how the different phases of the USDP process are related.

1.4 Definitions, Acronyms, and Abbreviations

CT – Cost & Unit Test.
CVM – **Communication** Virtual Machine.
DD – Detailed Design.
IT – Integration Test.
NCB – Network Communication Broker.
PD – Product Design.
QoS – Quality of Service.
RQ – Requirements.
SE – Name of the overlaying layer that sits atop UCM, Synthesis Engine.
UCM – User-Centric Communication Middleware.

1.5 Overview of Document

The rest of this document consists of more detailed information about the development process of UCM. This is broken down into seven more chapters. **Chapter 2** descirbes the current system. **Chapter 3** contains the project schedule and tasks. **Chapter 4** introduces the system's use cases with both functional and nonfunctional requirements. **Chapter 5** describes the proposed system architecture along with the subsystem decomposition, hardware and software mapping, and persistent data management. **Chapter 6** contins the models produced during the object design phase. **Chapter 7** contains the test cases and the results of the tests made to to the system. **Chapter 8** includes a glossary of terms used in this document for the

general reader. **Chapter 9** contains a page with the signature of the team members. C**hapter 10** contains the appendices consisting of supporting documentation and visual aids for the previous chapters. **Appendix A** contains the project schedule. **Appendix B** contains the use cases. **Appendix C** contains user interfaces, in the case of this system this is the API of NCB. **Appendix D** has the detailed class diagrams of the system. **Appendix E** contains class interfacesof all the subsytems implemented. **Appendix F** contains the documented test drivers. **Appendix G** contains dairy of meeting tasks, in these all the meetings and tasks assigned are described.

2. Current Systems

The User-Centric Communication Middleware (UCM) is part of the CVM architecture, which is supposed to take control scripts, generated from the synthesis engine and execute them by invoking the APIs provided by NCB layer. However, UCM is not properly implemented in the current CVM. With the current prototype, Synthesis Engine directly interacts with NCB, bypassing the UCM layer, decreasing both system flexibility and extensibility.

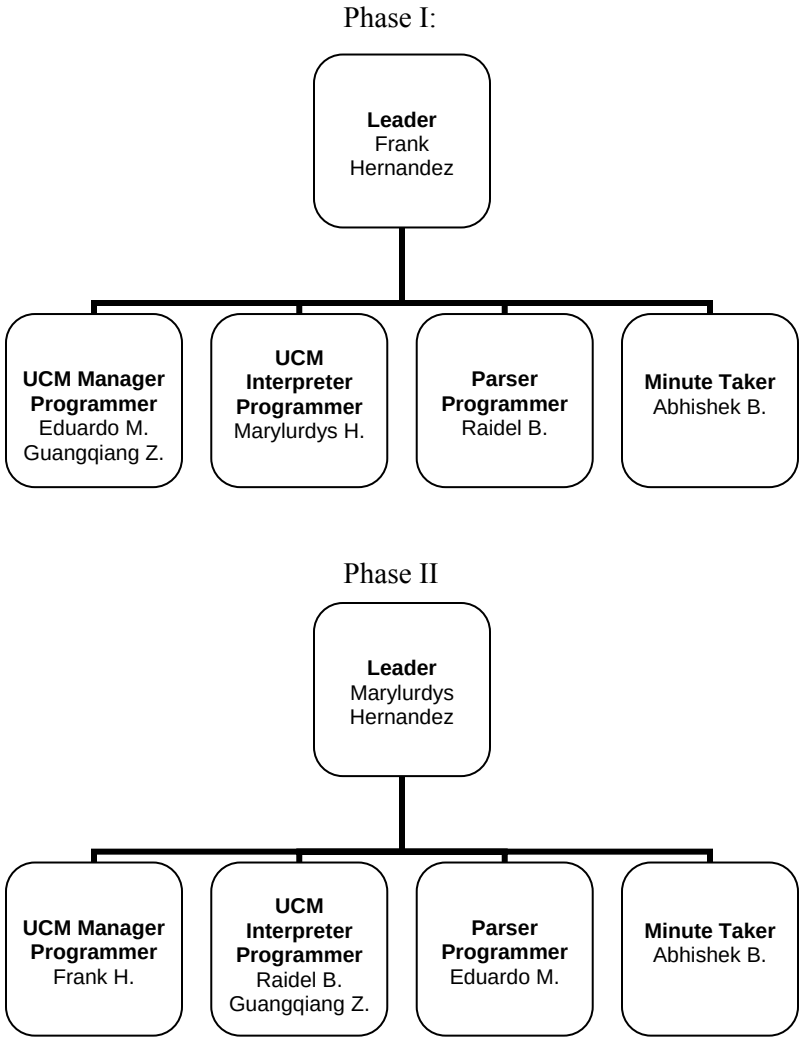
Another problem with the current system is that the local repository facility is not actually fully implemented. Storing schemas, and macros and other runtime information are not supported by any layers of the prototype, resulting in disobedience of the original motivation of the CVM paradigm: to provide QoS communication services with great ease and flexibility.

3. Project Plan

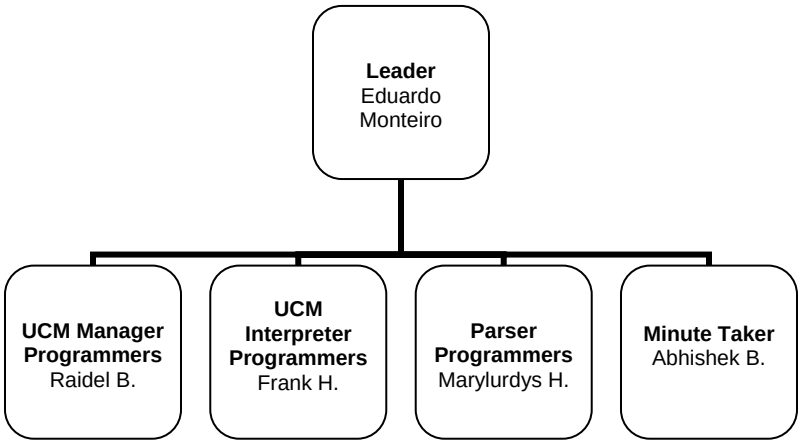
This section discusses the fundamental structure of the User-Centric Communication Middleware (UCM for short) development plan. This includes the different roles involved in creating this project, the hardware and software requirements important to its functionality and maintenance, and the milestones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

3.1 Project Organization

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



Phase III



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.

Parse Programmer – Handles the design of the structure of the UCM Control Script parser. Will also handle the main concerns with the implementation of the parser.

UCM Manager Programmer – Handles the implementation of the UCM_Manager Subsystem. Has the responsibility of programming all of the functionality of the UCM_Manager.

UCM Interpreter Programmer – Handles the implementation of the UCM_Interpreter Subsystem. Has the responsibility of programming all of the functionality of the UCM_Interpreter.

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.

3.2 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. Rational Rose
3. StarUML
4. Microsoft Word
5. Microsoft Visio
6. Microsoft Project
7. Microsoft PowerPoint
8. Eclipse 3.2
9. Costar 7.0 Demo
10. Janino library

3.3 Work Breakdown

Tasks and Milestones

The tasks in the development of UCM 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 completion of Design Phase and started on the design and implementation of the microkernel of the UCM Interpreter. This included the implementation of the control script parser. 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 A for project schedule and Appendix B for diary.

ID	Task Name	Duration	Predecessors
1	Identify Requirements	4 days	
2	Write use Cases	1 day	1
3	Validate Use Cases	7 days	2
4	Create Use Case Models	2 days	1
5	Create Object Diagrams	9 days	1
6	Create Sequence Diagrams	12 days	2
7	Create System Use Case Model Presentation	3 days	1
8	Draw Gantt Chart	1 day	7,6,5,4,3
9	**Project Deliverable 1**	0 days	8,7,6,5,4,3,2,1
10	Subsystem Decomposition	7 days	
11	Create Interface Mockups	6 days	
12	Write Parsers and transformers	11 days	
13	Write Macros	13 days	
14	Create calls to SE	4 days	
15	Create calls to NCB	4 days	
16	Generate Code based on models	14 days	14,15
17	**Project Deliverable 2**	0 days	10,11,12,13,14,15,16,9
18	System Tests	4 days	
19	SubSystem Tests	5 days	
20	Evaluation of Tests	3 days	18
21	Final Presentations	5 days	
22	**Final Project Deliverable**	0 days	18,19,20,21,17

3.4 Project Cost Estimate

The cost for the project was calculated via the use of COCOMO II model for project cost calculation. The tool that was used was Costa 7.0 Demo. This was selected for the large amount of detail and information that is generated as well as its usability. Bellow are the tables that were found more relevant to the cost estimate calculation. The cost model used in this calculations is $PM_{nominal} = A \times (Size)^{B}$ where PM – Person Months, A – multiplicative effects on effort with projects of increasing size, B – accounts for the relative economies or diseconomies of scale encountered for s/w projects of different sizes $B = 0.91 \times 0.01 \times \sum w_i$ and w_i - scale drivers.

UCM_Estimate - Cost & Breakage Report

Costar 7.0 Demo 09/19/2007 01:12:25 Page: 1

Estimate Name:	UCM_Estimate	Estimate ID:	
Model Name:	COCOMO II 2000	Model ID:	2000
Process Model:	COCOMO II Model	Phases:	Waterfall

Increment 1 of 1

Names of Leaf Components	Developed Size	RQ Cost	PD Cost	DD Cost	CT Cost	IT Cost	Total Cost (K$)
UCM	5,000	5.9	17.9	28.8	35.9	17.7	106.3
Incr 1 Total	5,000	5.9	17.9	28.8	35.9	17.7	106.3
Grand Total	5,000	5.9	17.9	28.8	35.9	17.7	106.3

Fig 3.4.1 UCM_Estimate Cost Breakdown

Description: This table shows the breakdown of the cost among the stages of the software creation. The stages are: RQ – Requirements, PD – Product Design, DD – Detailed Design, CT – Code & Unit Test and IT – Integration Test.

UCM_Estimate - Cost Driver Report

Costar 7.0 Demo 09/19/2007 01:07:27 Page: 1

Estimate Name: UCM_Estimate
Model Name: COCOMO II 2000
Process Model: COCOMO II Model

Estimate ID:
Model ID: 2000
Phases: Waterfall

Component Name	EAF	ACAP	APEX	CPLX	DATA	PCAP	RELY	RUSE	SCED	STOR	TIME	TOOL	PLEX	LTEX	PCON	SITE	PVOL	DOCU	
UCM	1.1059	H	VL	VH	H	H	N	XH	N	N	N	N	H	H	N	XH	L	H	

Fig 3.4.2 UCM_Estimate Cost Driver

Description: This table shows the cost drivers affecting the cost of our system. For specific meaning of each of the divers please refer to the COCOMO II specifications and manual.

UCM_Estimate - Effort Report

Costar 7.0 Demo 09/19/2007 01:12:56 Page: 1

Estimate Name: UCM_Estimate
Model Name: COCOMO II 2000
Process Model: COCOMO II Model

Estimate ID:
Model ID: 2000
Phases: Waterfall

Effort per Component (Person-Months)

Component Name	RQ	PD	DD	CT	IT	Total RQ to IT
UCM	1.4	3.3	5.1	6.9	4.0	20.6

Effort Summary

Component Totals	1.4	3.3	5.1	6.9	4.0	20.6
Grand Total	1.4	3.3	5.1	6.9	4.0	20.6

Fig 3.4.3 UCM_Estimate Effort

Description: This table shows the effort report for our system. Effort is calculated in P/M or Person-Months required for this project.

4. Requirements 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 follows a vertical development approach where each application is built on top of a low-level network abstraction. This development approach is what drives the cost of developing this application up. Also is this approach that slows down the pace of innovation of the new generation communication applications. The User-Centric Communication Middleware aims at solving this problem. The UCM aims to provide unified higher-level abstraction for the class of multimedia communication applications.

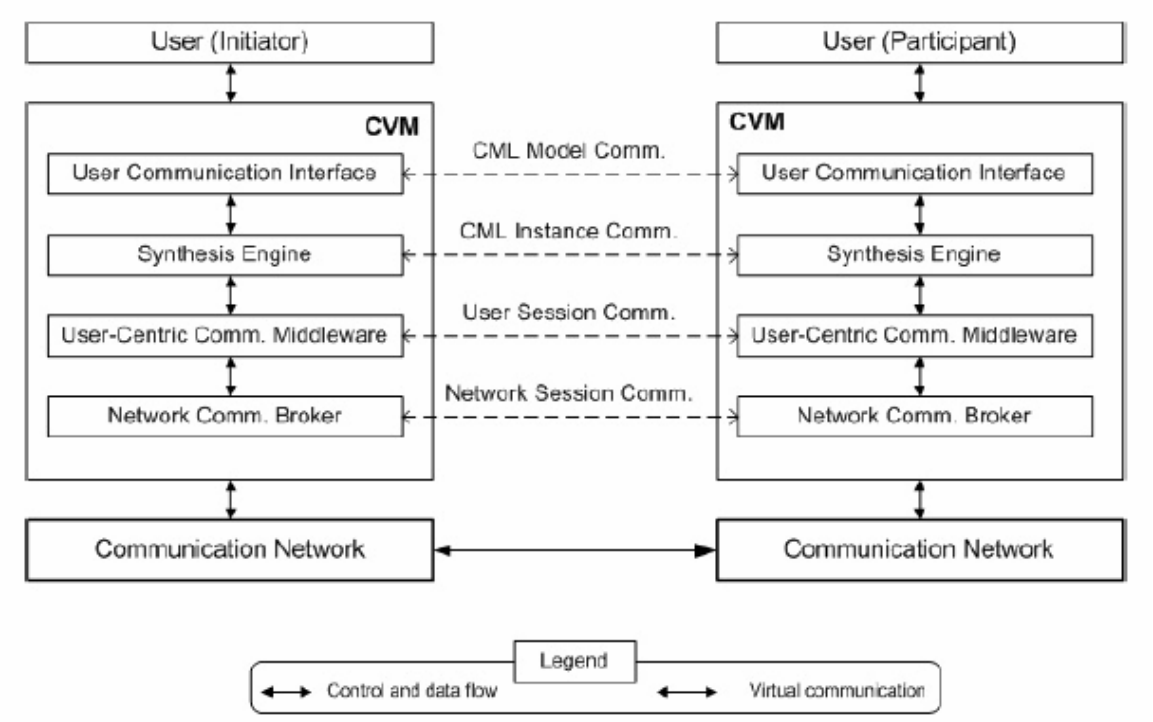


Fig. 4.1.1 CVM Design

Description: This is a representation of the CVM system. From this system we will be implementing the UCM layer. This layer will handle the communication between the Synthesis Engine and the Network Communication Broker.

4.1 Functional and Nonfunctional Requirements

4.1.1 Overview

The UCM system will practically allow a user to create communication applications without having to worry about creating the network protocol handlers every time. The system will lie under the Synthesis Engine, and over the Network Communication Broker. The system will then interpret the UCM control scripts output by the Synthesis Engine; it will log these control events, and then call NCB-specific commands. UCM will provide unified higher-level abstraction for the class of multimedia communication applications

4.1.2 Functional Requirements

The system shall:

1. Parse a control script sent from the Synthesis Engine. Refer to use cases UCM_01-24 in Appendix B.
2. Find the macro for each control command in the control script. Refer to use cases UCM_01-24 in Appendix B.
3. Notify the Synthesis Engine of any macro in the control script that is not defined in the repository. Refer to use cases UCM_01-24 in Appendix B.
4. Execute each command macro and call the underlying NCB. Refer to use cases UCM_01-24 in Appendix B.
5. Notify the Synthesis Engine of events relevant to this layer. Refer to use cases UCM_01-24 in Appendix B.
6. The possible control command supported by the system are:
    a. Login
    b. Logout
    c. CreateConnection
    d. DeclineConnection
    e. SendMedia
    f. AddParticipants
    g. RemoveParticipants

Note: See Appendix B for use cases UCM_01-UCM_07 and UCM_09

4.1.3 Non-Functional Requirements

1. Usability

a. System users are other subsystems(NCB and SE), which communicate through well-defined scripts and interfaces that are easy to operate for other systems.
b. This use case should provide nice predefined error messages to the user.

2. Reliability
    a. 5%-10% failure rate for every 24 hours of use.

3. Performance
    a. Requests should be handled in less than 2 minutes, if no other requests exist. Response time can be more depending on the number of participants in the active connection.
    b. The total process for searching the macro definition should not be more than 5 seconds.

4. Supportability
    a. The command must be properly supported by the Event handler of UCM
    b. The 15 exceptions that might result from the runtime execution must be handled.
5. Implementation
    a. Must be implemented in Java.

4.2 Use case diagrams.

The actors for UCM are the NCB and the SE (see Appendix 8). The SE passes a control script down to the UCM by means of the executeScript() call. The UCM then parses this scripts and performs class the NCB to satisfy the request from SE. NCB then signals back the success or failure of this call to UCM via events.

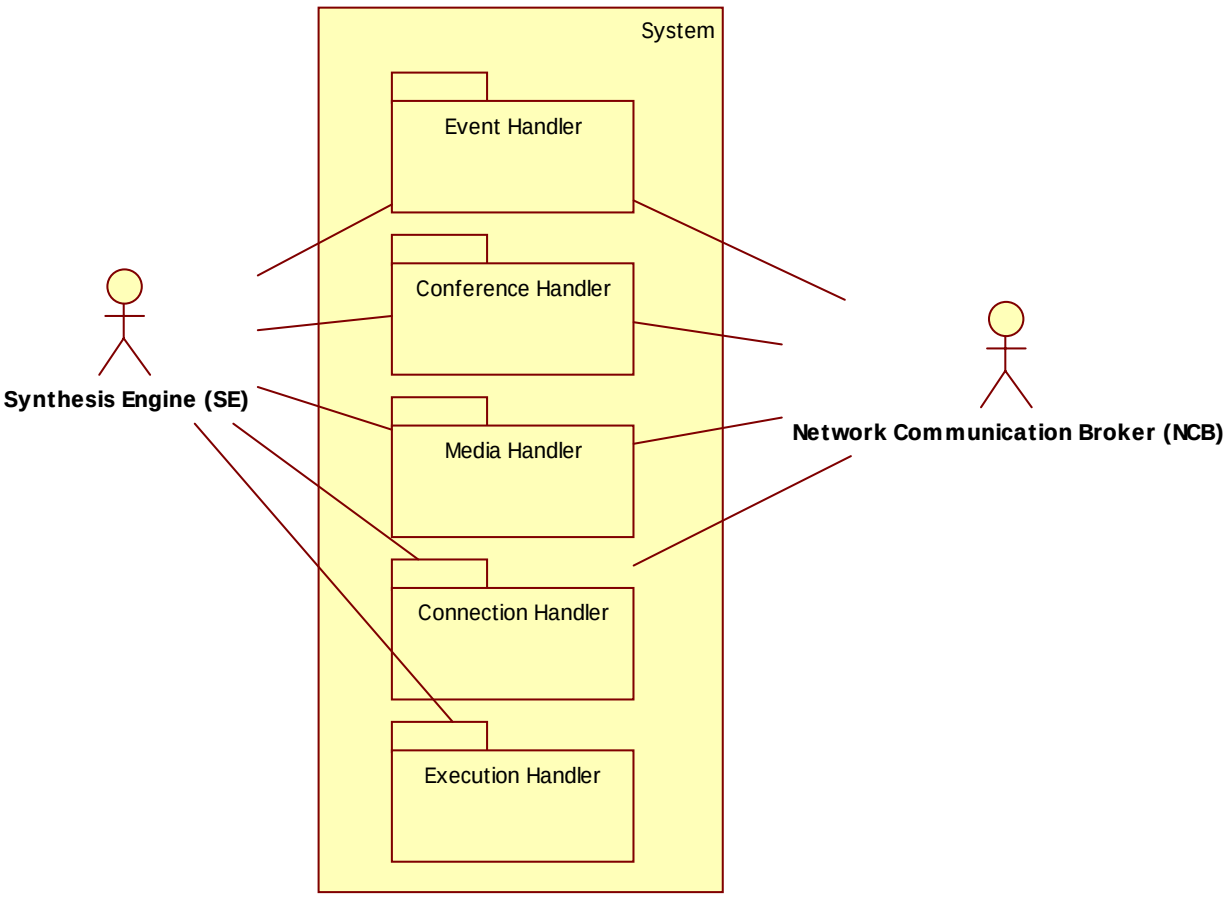


Fig. A.1 Use Case Model Design

Description: This is the use case model for the UCM system. This is a representation all of the possible interactions that both actors SE and NCB can have with the system. Packages are used to simplify the diagram.

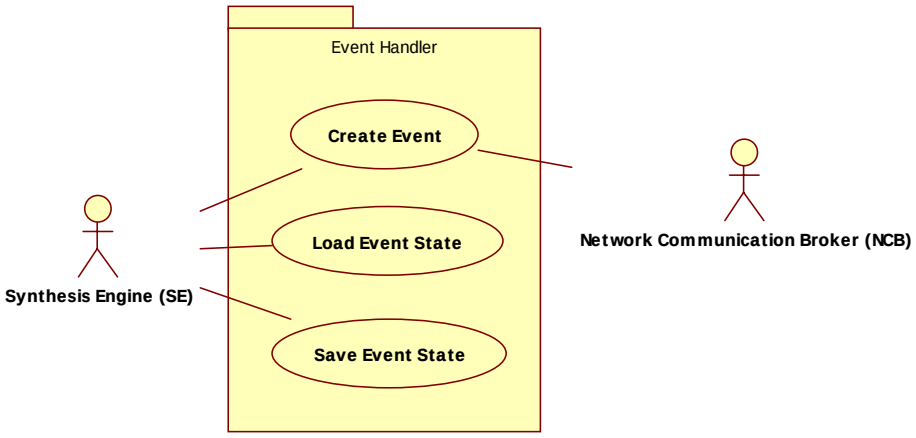


Fig. A.2 Use Case Model – Event Handler Package

Description: This is the use case model of the Event Handler package for the UCM system. These use cases will occur anytime that there is any kind of event that need handling. These events will range from basic execution to mere notifications of actions.

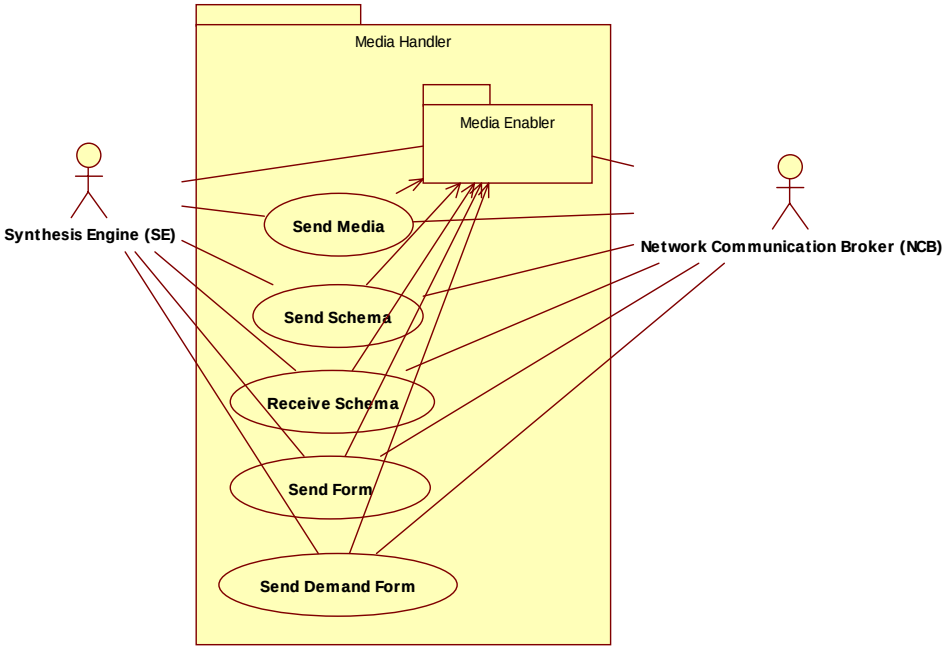


Fig. A.3 Use Case Model – Media Handler Package

Description: This is the use case model of the Media Handler package for the UCM system. These use cases will occur anytime that the system attempts to initialize any kind of media operation. These operations include the sending of files, schema, and even forms, among others.

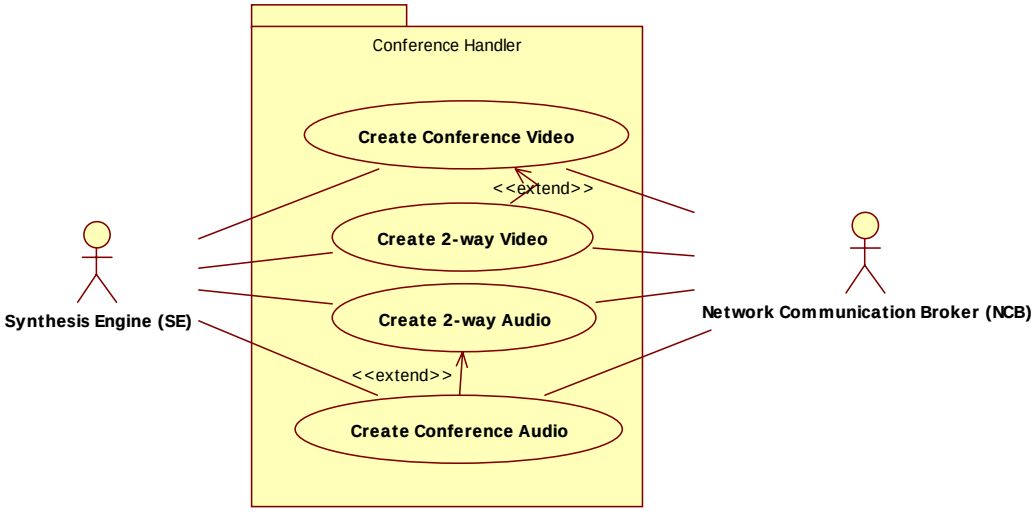


Fig. A.4 Use Case Model – Conference Handler Package

Description: This is the use case model of the Conference Handler package for the UCM system. These use cases will occur anytime that there is any kind of conference creation. This use cases represent how UCM handles the creation of basic conference protocols.

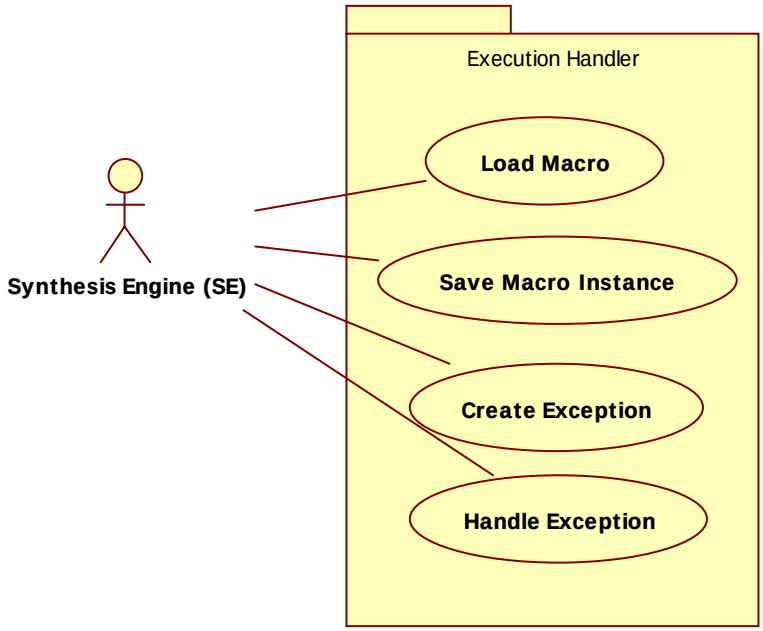


Fig. A.5 Use Case Model – Execution Handler Package

Description: This is the use case model of the Execution Handler package for the UCM system. These use cases will occur anytime that there are any user initiations in the system.

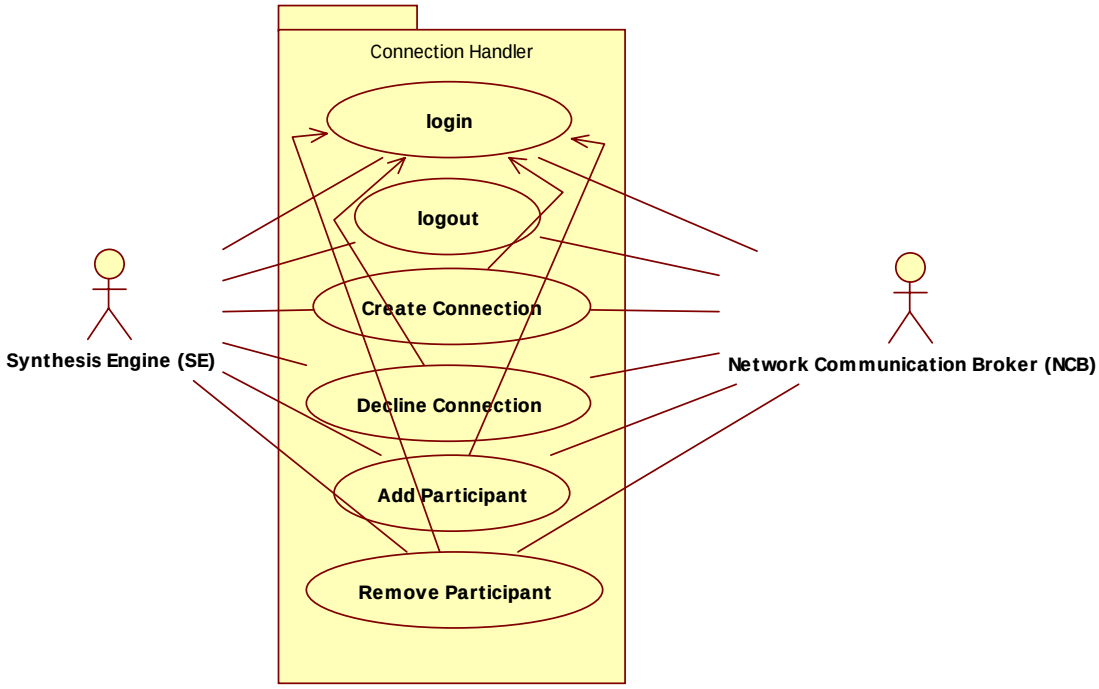


Fig. A.6 Use Case Model – Connection Handler Package

Description: This is the use case model of the Connection Handler package for the UCM system. These use cases represent the possible steps taken by the system when handling the basic connection issues. Such basic connection issues include, login into the system, log out, and create connection and so on.

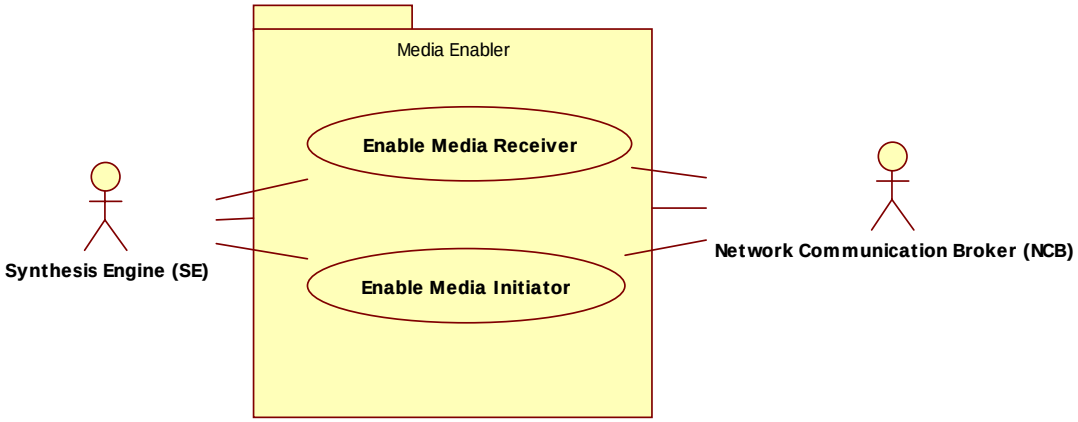


Fig. A.7 Use Case Model – Media Enabler Package

Description: This is the use case model of the Media Enabler package for the UCM system. This package was created to abstract the Media Handler Package.

4.3 Requirement Analysis

The Requirements Analysis was partly performed by extracting requirements from Researchers involved in the CVM project. System details discussed in the CVM meetings were incorporated into the development of the UCM layer. Additional requirements were obtained from documents and publications that were produced outside of class by the group responsible for the CVM project. Examples of such documents are "CVM - A Communication Virtual Machine" by Deng et al, "UCM State Machine", and "UCM Control Script". Any assumptions were verified with the professor and the parties involved in the CVM project.

5. Proposed Software Architecture

The UCM layer 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. 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.

5.1 Overview – Package Diagram

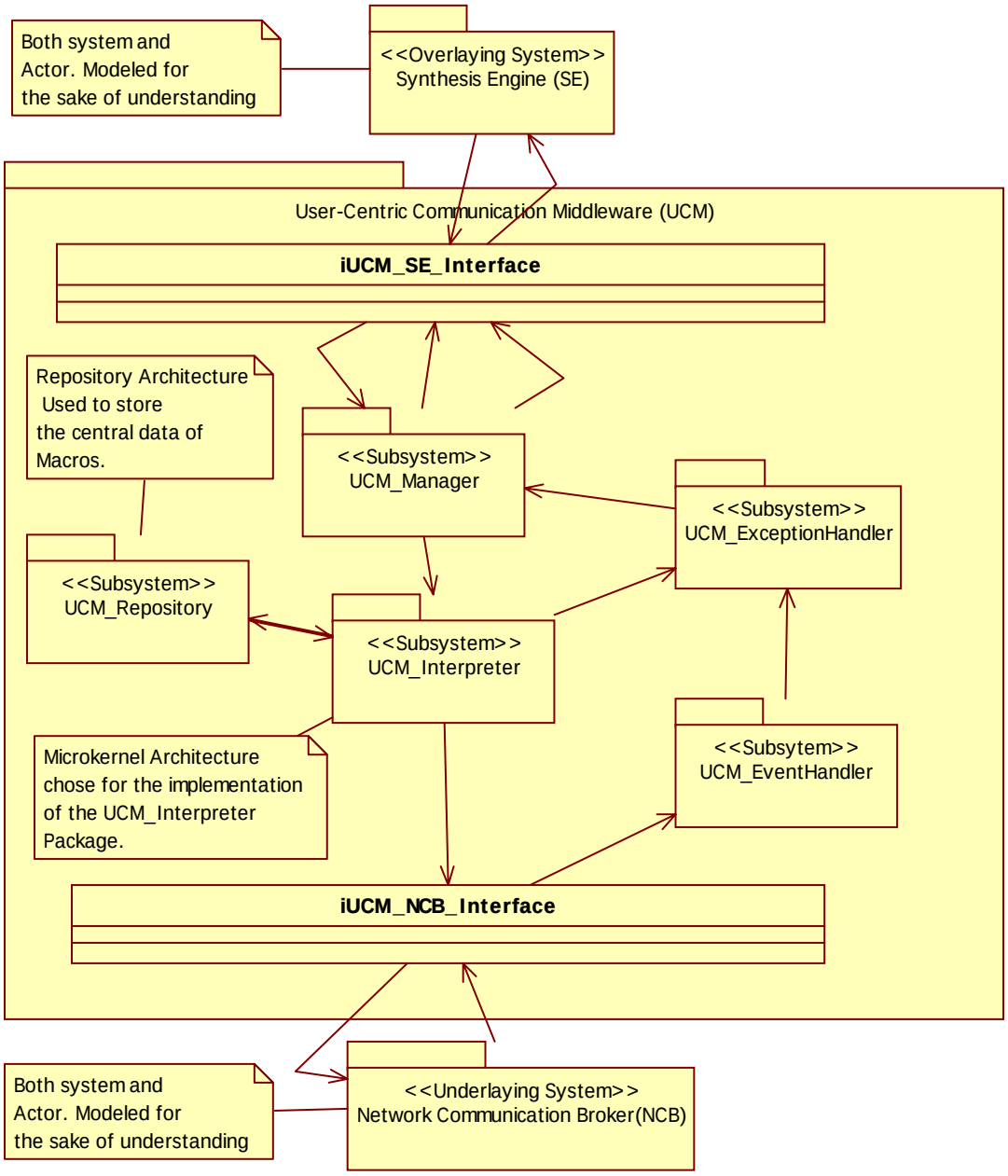


Fig. 5.1 UCM Architecture
Description: This is the package diagram for UCM. It displays the subsystem decomposition of the architecture.

Microkernel Architecture: Our project uses the Microkernel Architectural style to enclose the functionality of the UCM Interpreter as a subsystem that coordinates and manages the global control flow of the UCM framework. It allows and manages the access to the core functionality of the system by providing a complete interface to the other subsystems,

including those that serve as bridges to the layers above and below UCM, the Exception Handler and the Repository. By enclosing the core functionality, it also serves as a socket that allows plugging in future desired functionality into the system.

Repository Architecture: The UCM Repository, the subsystem that contains the macros with the source code to execute the different functions allowed by the system and specified in the use cases, is designed using the Repository Architectural style. The UCM Repository is a single data structure whose concurrency and integrity issues management is facilitated through this architectural style.

5.2 Subsystem Decomposition

Subsystems:

UCM_Manager: UCM_Manager coordinates the activities of UCM. The UCM_Manager will delegate the script received to the UCM_Interpreter. Also it will notify the SE of any events or exceptions that must be known by it. UCM_01 – UCM_24.

UCM_Repository: UCM_Repository stores the macros for the execution of the control scripts. Macros can be added at any time. This ensures the extensibility of the application without having to write or change any core code. Related use cases: UCM_01 – UCM_24.

UCM_Interpreter: UCM_Interpreter parses and interprets the control scripts, loads macros from the repository and makes a sequence of calls to the NCB to realize the communication described in the control script. Related use cases: UCM_01 – UCM_24.

UCM_ExceptionHandler: The UCM_ExceptionHandler will be responsible for deciding how to act on exceptions received due to control script faults, NCB specific messages, or bad function call returns. Related use cases: UCM_19 and UCM_20.

UCM_EventHandler: The UCM_EventHandler will coordinate and orchestrate the events raised by other subsystems as well as deciding what to do in each case. Related use cases: UCM_21 – UCM_23.

Synthesis Engine (System/Actor): The Synthesis Engine is composed of a suite of algorithms that automatically synthesize a user communication schema instance to an executable form called communication control script. Negotiates the schema among participants of a communication to ensure that all parties agree to a consistent schema. Automatically transforms this schema.

Network Communication Broker (System/Actor): The Network Communication Broker provides a network-independent API to the UCM and works with the underlying network protocols to deliver the communication services. Utilizes and coordinates the available, low-level network and hardware services. Provides self-management in response to dynamics of the underlying infrastructure.

5.3 Hardware and Software Mapping

UCM - The UCM software and hardware requirements are highly flexible. While most communication systems are Operating System dependent, the UCM only needs a Java Virtual Machine to run. Since Java can be installed on a wide array of devices ranging from high performance servers to remote controls and kitchen ovens, the hardware running under the Java Virtual Machine is irrelevant. The only requirements are a network connection, input device, speakers and microphone.

Repository - The repository that stores the macros to be executed by the UCM is platform independent. It abstracts the storage used by maintaining general storage and retrieval methods that work for databases or file systems. Other types of storage systems may be used by creating additional classes that interact with the Repository class and create an interface for the underlying storage method.

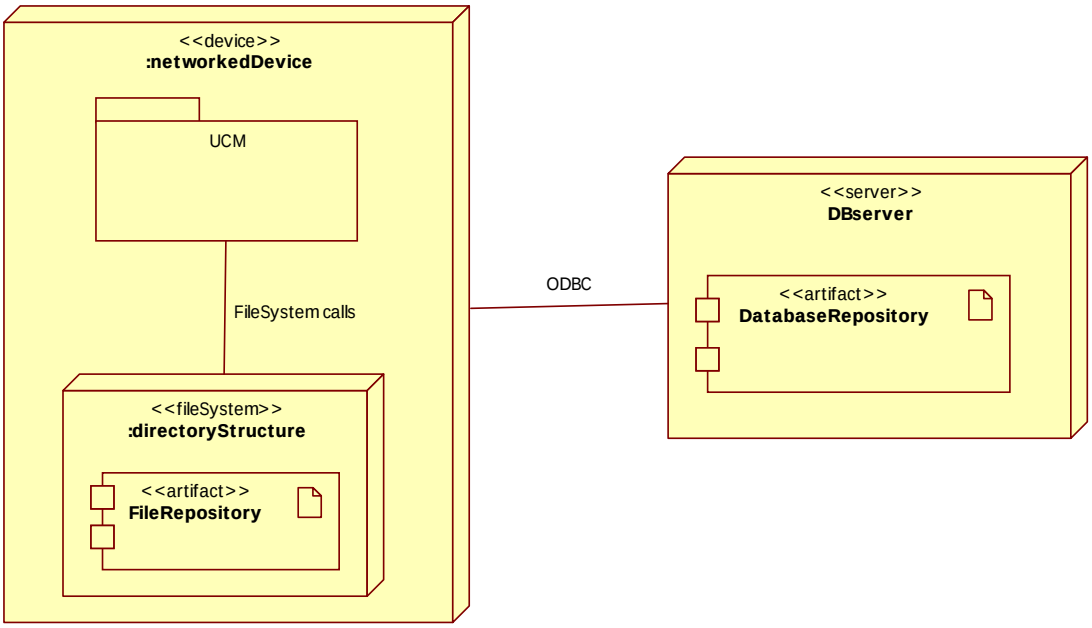


Fig 5.3.1 Deployment Diagram

Description: This is the deployment diagram for the UCM system.

5.4 Persistent Data Management

The persistent data identified for the UCM framework at this stage of development consist of the macro information for those functions recognized in the use cases. The data stored in the repository, through a Relational Database Management System, contains a table called Macros whose schema is defined as follows:

Macros {*name:string, returnType:string, paramTypeList:string, paramNameList:string, script:string*}

Where *name* is the name of the function, *returnType* is the type of the object returned by the function, *paramTypeList* is the list of the type of each parameter that has passed as the argument to the function, *paramNameList* is the name of each parameter, and *script* is the actual source code that contains the functionality of the function.

In the case of the *paramTypeList* and the *paramNameList*, string objects are retrieved from the database and parsed into Array Lists to pass as parameters of the *Macro* object constructed by the repository and returned to the UCM Interpreter.

6. Object Design

This section covers 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 UCM. 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.

6.1 Overview – Minimal Class Diagram

6.1.1 Minimal Class Diagram

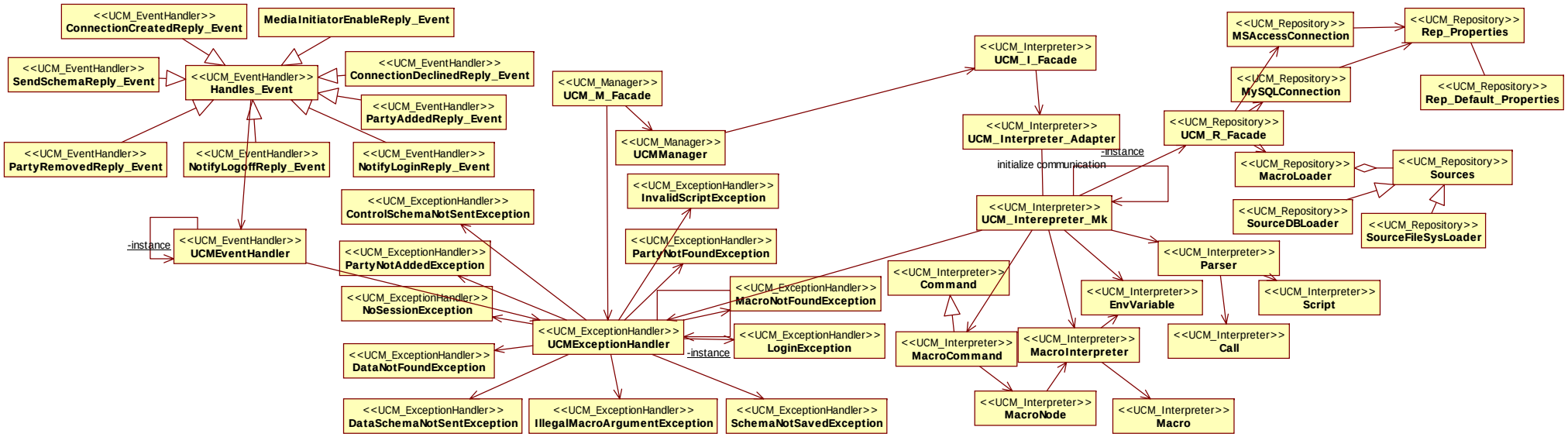


Fig 6.1.1.1 Minimal Class Diagram

Description: This is the minimal class diagram of UCM. Show class interaction between subsystems.

6.1.2 Class Description

UCM_M_Façade: Façade class that provides an interface into the UCM_Manager subsystem. The purpose of this class is to aid future expansion of this subsystem.

UCMManager: Controls the workflow inside the UCM system. This class also notifies the overlying system of any event that are specific to it.

UCM_I_Façade: Façade class that provides an interface into the UCM_Interpreter subsystem. The purpose of this class is to aid future expansion of this subsystem.

UCM_Interepreter_Mk: This is the microkernel as in the architecture pattern. This class control the parsing of scripts, the loading of macros, and the queuing and execution of commands.

UCM_Interpreter_Adapter: Adapter class as in the microkernel architecture pattern. It makes the parse script requests to the UCM_Interepreter_Mk class.

Command (Interface): This is the Command Design Pattern interface.

MacroCommand: This class handles encapsulation of the execution of macros inside our system. This is part of the incorporation of the command design pattern into our system.

MacroNode: This class encapsulates the script editor and the parameters needed to execute a macro. This node is then encapsulated by the MacroCommand class. This is the 'Receiver' of the command design pattern.

Macro: This class encapsulates all the data specific to any macro. This object simplifies the transfer of information across subsystems dealing with macros.

MacroInterpreter: MacroInterpreter encapsulates the creation of a script evaluator object which will be executed using Janino inside the MacroCommand execute.

UCMEventHandler: This class handles all the possible events that the NCB layer fires.

ConnectionCreatedReply_Event: This class encapsulates the event source and the reply messages of invoking *createSession* coming from NCB into a single event.

NotifyLoginReply_Event: This class encapsulates the event source and the reply messages of invoking *login* method coming from NCB into a single event.

SendSchemaReply_Event: This class encapsulates the event source and the reply messages of invoking *sendSchema* coming from NCB into a single event.

PartyAddedReply_Event: This class encapsulates the event source and the reply messages of invoking *addParty* coming from NCB into a single event.

PartyRemovedReply_Event: This class encapsulates the event source and the reply messages of invoking *removeParty* coming from NCB into a single event.

UCMExceptionHandler: This class handles the possible exceptions resulting in the runtime of the script interpretation process.

MacroNotFoundException: This class defines the exception thrown by the macro loader when the macro is not found in the local repository.

IllegalArgumentException: This class defines the exception thrown by the macro Interpreter when the macro has an unexpected argument passed to it.

PartyNotFoundException: This class defines the exception thrown by the UCM Interpreter when it is trying to remove a party that is not in the current connection.

NoSessionException: This class defines the exception thrown by the UCM Interpreter when it is trying to add participants or send data in the connection that has no corresponding session in the NCB layer.

DataNotFoundException: This class defines the exception thrown by the UCM Interpreter when it is trying to send data that could not be found locally or on the network.

InvalidScriptException: This class defines the exception thrown by the UCM Interpreter when it is trying to pass the string received from Synthesis Engine.

UCM_R_Façade: Façade class that provides an interface into the UCM_Repository subsystem. The purpose of this class is to aid future expansion of this subsystem.

Sources: Interface to Source loaders.

MacroLoader: The MacroLoader class retrieves, from a given Source (a Database, a File System, etc), the information necessary and creates a Macro.

SourceDBLoader: This class retrieves a Macro from a database given a function name. There is no overloading of a function name.

SourceFileSysLoader: This class retrieves a Macro from a file system given a function name.

Parser: This class parses the control scripts.

Script: Data structure that holds a control script in memory.

Call: Call holds function calls in a given control script.

6.2 State Machine

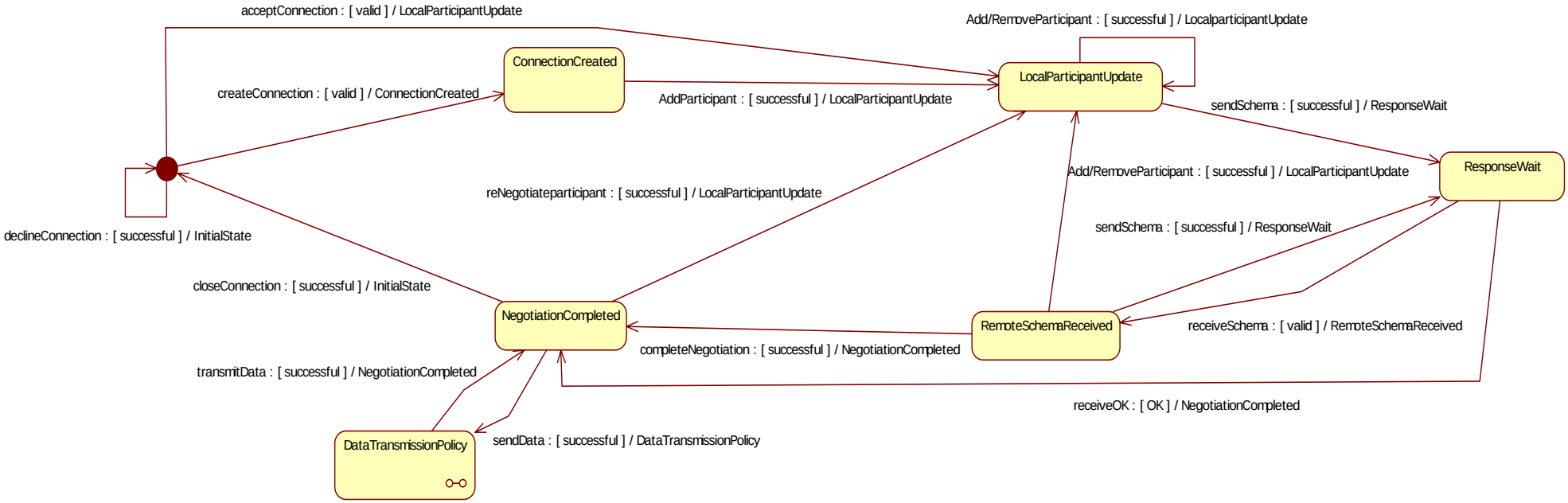


Fig 6.2.1 UCM State Machine

Description: This state machine represents all the states in which the UCM will be when establishing communications. A connection is first created, then every participant is added. Once all the participants are added the schema is sent, and when the negociation is competed the transmission policy is sent.

6.3 Object Interaction

This section contains the redefined version of the sequence diagram first created for the requirement phase. These sequence diagrams are now more specific and represent the interaction between objects. These objects are now more closely related to the classes that will be implemented.

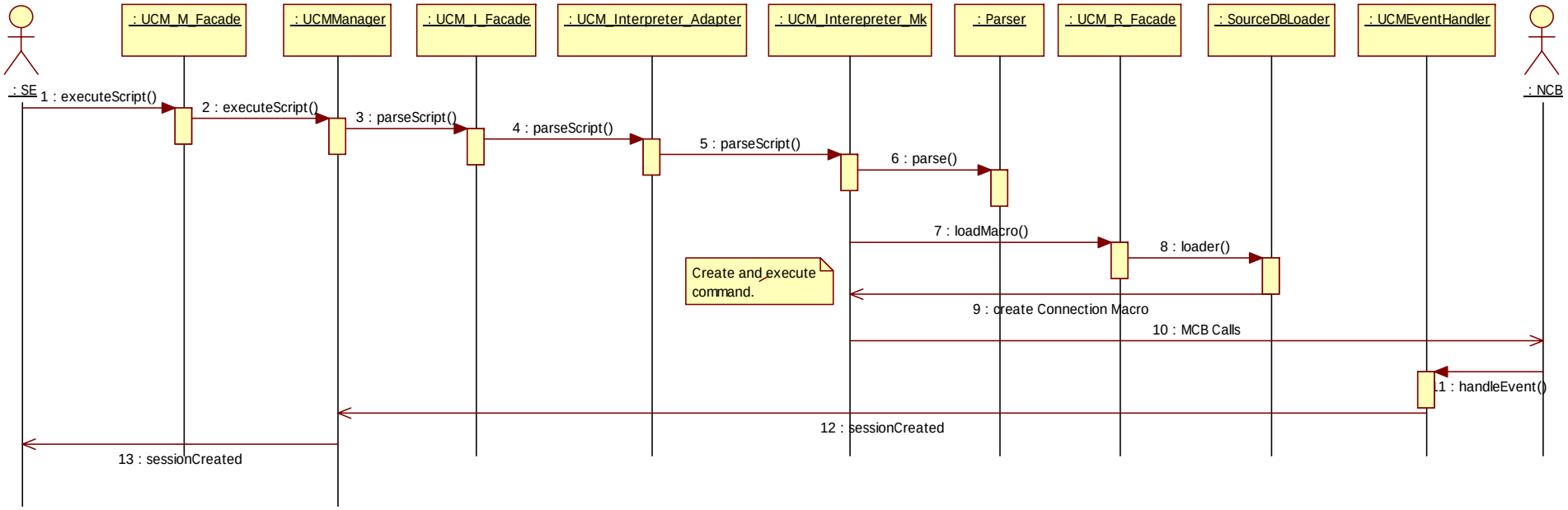


Fig 6.3.1 Sequence Diagram - Create Connection

Description: This sequence diagram represents the steps that UCM takes for handling a control script containing the 'CreateConnection' command from SE. A control script is received from the Synthesis Engine, by the manager, then the manager delegates it to the interpreter which then parses it. Once the script is parsed the macro for the execution of the 'CreateConnection' command is loaded, and initialized by the values passed in the script. Finally it is executed to make calls from the Network Communication Broker to the to the Synthesis Engine.

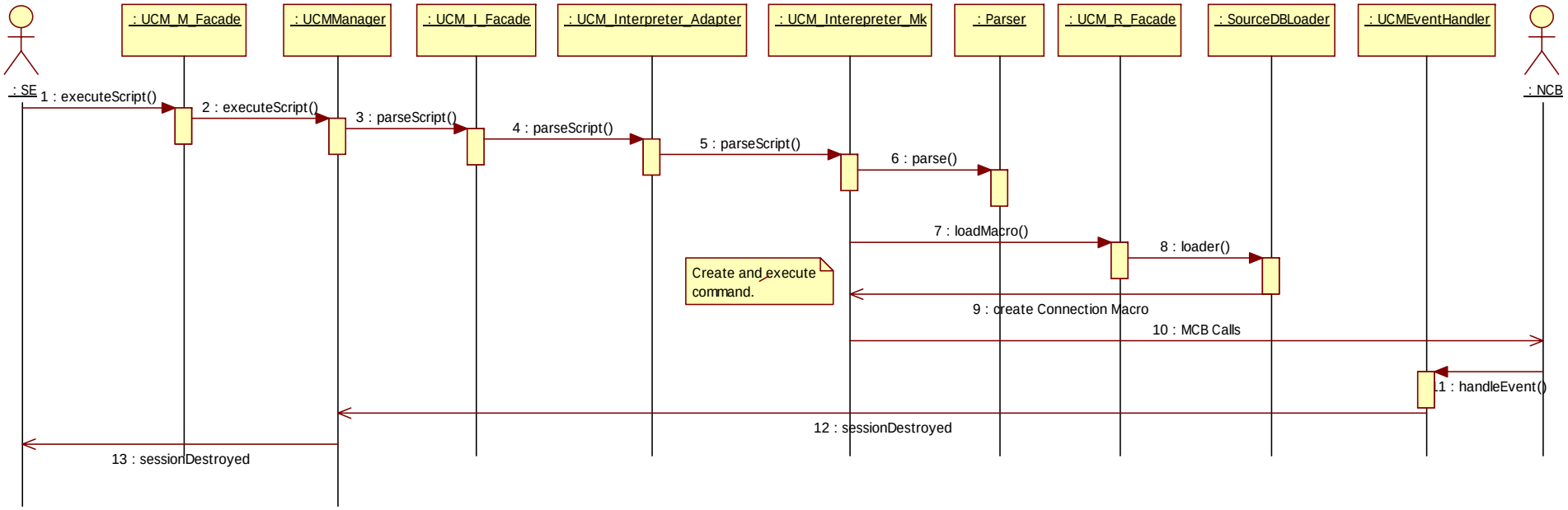


Fig. 6.3.2 Sequence Diagram - Destroy Connection

Description: This sequence diagram represents the steps that UCM takes for handling a control script containing the 'DestroyConnection' command from SE. A control script is received from the Synthesis Engine, by the manager, then the manager delegates it to the interpreter which then parses it. Once the script is parsed the macro for the execution of the 'DestroyConnection' command is loaded, and initialized by the values passed in the script. Finally it is executed to make calls from the Network Communication Broker to the Synthesis Engine.

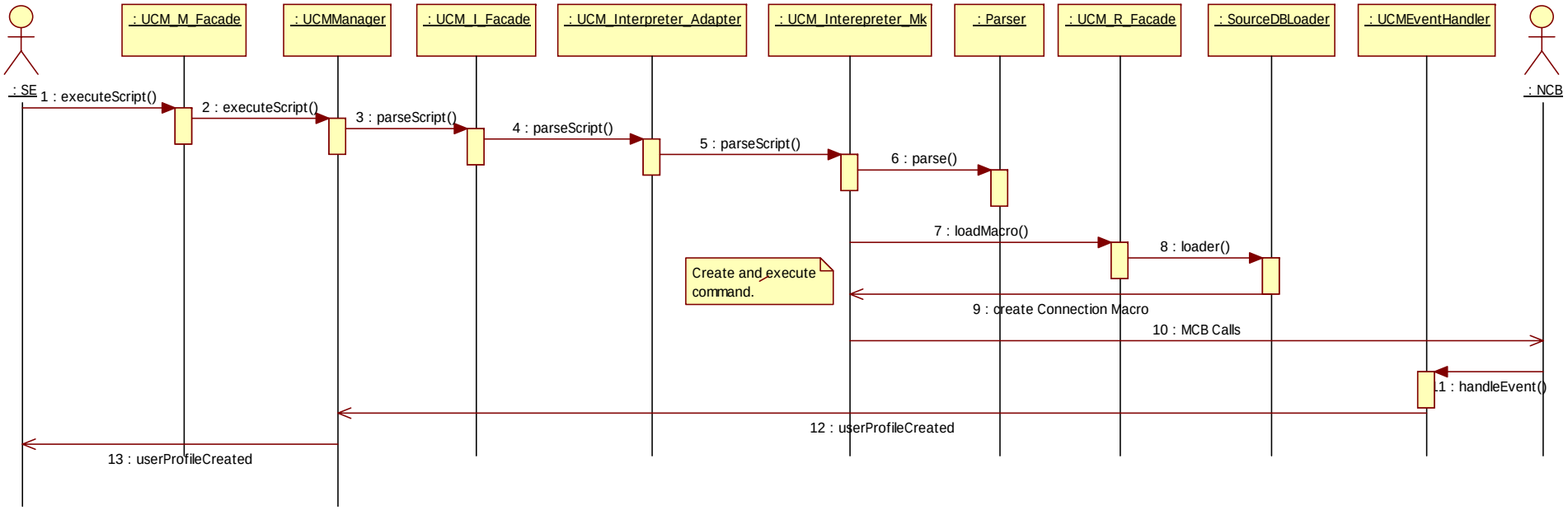


Fig. 6.3.3 Sequence Diagram - Login

Description: This sequence diagram represents the steps that UCM takes for handling a control script containing the 'login' command from SE. A control script is received from the Synthesis Engine, by the manager, then the manager delegates it to the interpreter which then parses it. Once the script is parsed the macro for the execution of the 'login' command is loaded, and initialized by the values passed in the script. Finally it is executed to make calls from the Network Communication Broker to the Synthesis Engine.

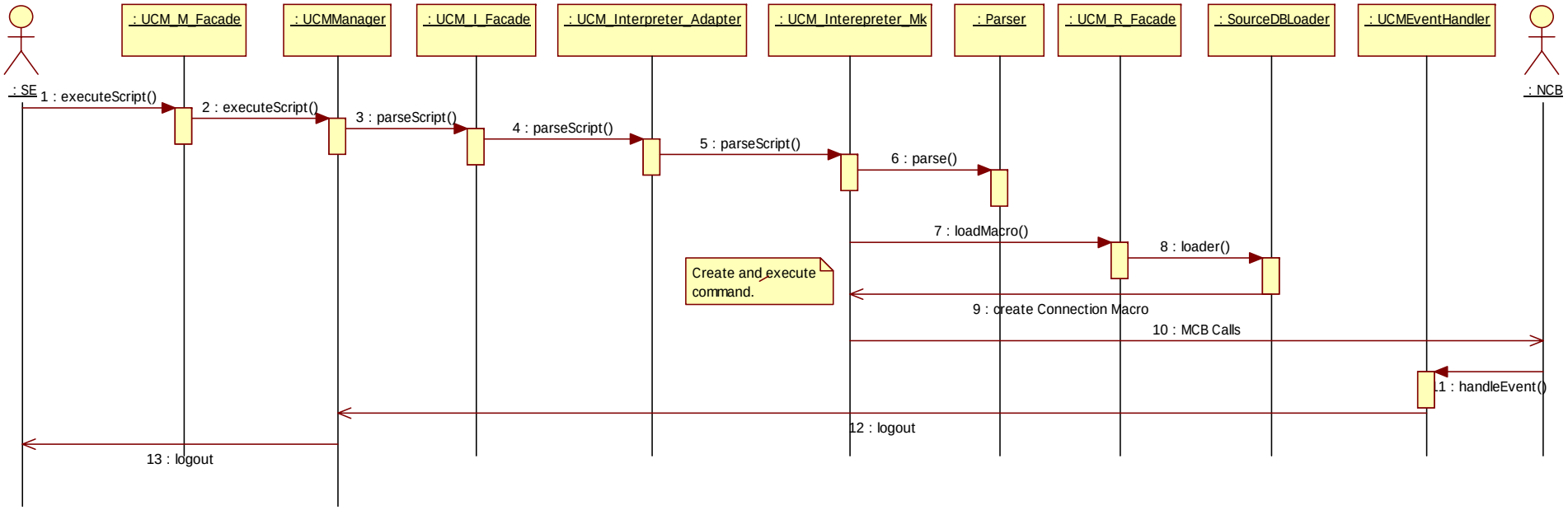


Fig. 6.3.4 Sequence Diagram - Logoff

Description: This sequence diagram represents the steps that UCM takes for handling a control script containing the ‘logoff’ command from SE. A control script is received from the Synthesis Engine, by the manager, then the manager delegates it to the interpreter which then parses it. Once the script is parsed the macro for the execution of the ‘logoff’ command is loaded, and initialized by the values passed in the script. Finally it is executed to make calls from Network Communication Broker to the Synthesis Engine.

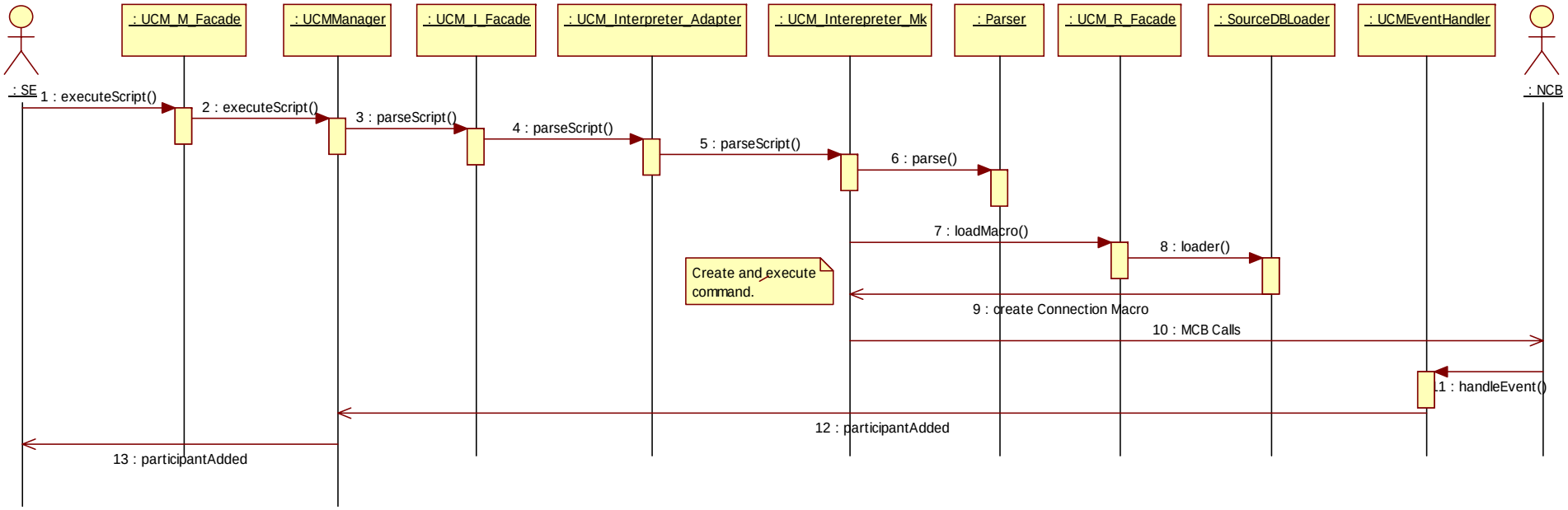


Fig. 6.3.5 Sequence Diagram – Add Participant

Description: This sequence diagram represents the steps that UCM takes for handling a control script containing the 'addParticipant' command from SE. A control script is received from the Synthesis Engine by the manager, then the manager delegates it to the interpreter which then parses it. Once the script is parsed the macro for the execution of the 'addParticipant' command is loaded, and initialized by the values passed in the script. Finally it is executed to make calls from Network Communication Broker to the Synthesis Engine.

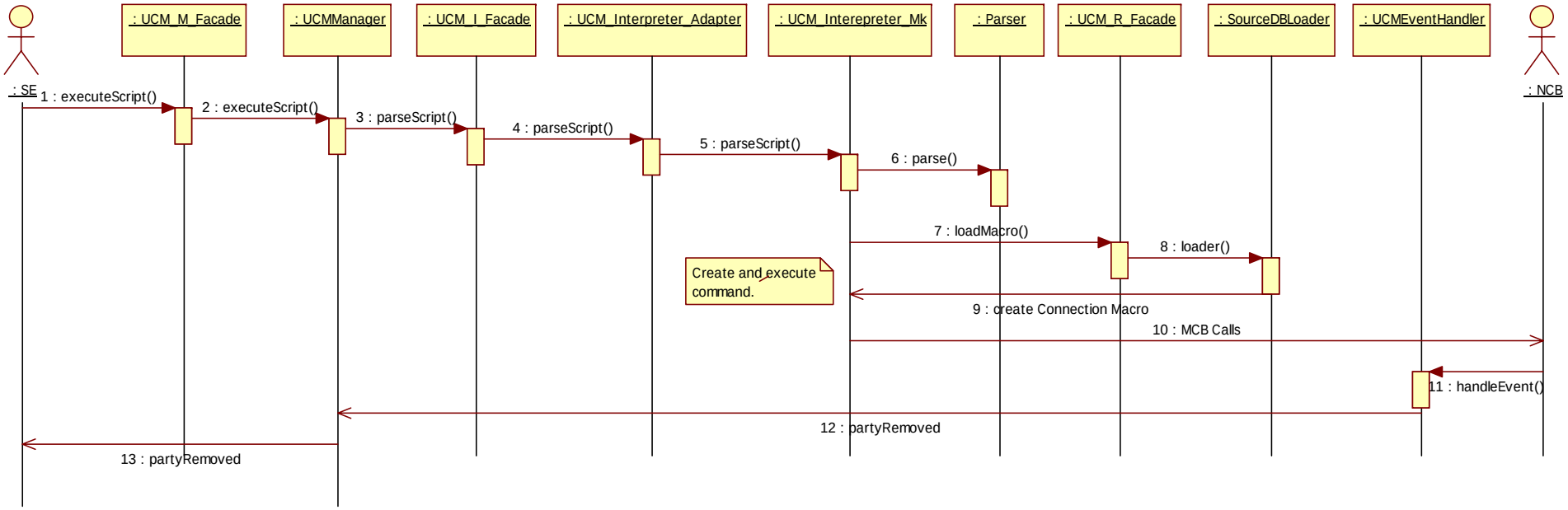


Fig. 6.3.6 Sequence Diagram – Remove Participant

Description: This sequence diagram represents the steps that UCM takes for handling a control script containing the 'removeParticipant' command from SE. A control script is received from the Synthesis Engine by the manager, then the manager delegates it to the interpreter which then parses it. Once the script is parsed the macro for the execution of the 'removeParticipant' command is loaded, and initialized by the values passed in the script. Finally it is executed to make calls from Network Communication Broker to the Synthesis Engine.

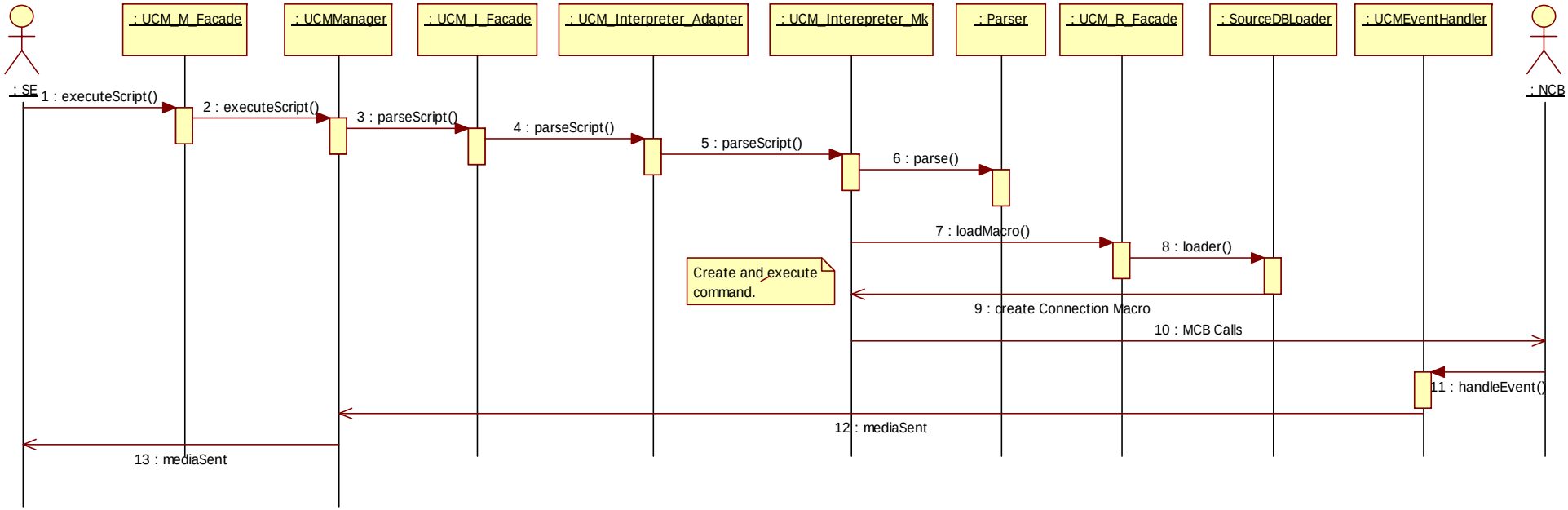


Fig. 6.3.7 Sequence Diagram – Send Media

Description: This sequence diagram represents the steps that UCM takes for handling a control script containing the 'sendMedia' command from SE. A control script is received from the Synthesis Engine by the manager, then the manager delegates it to the interpreter which then parses it. Once the script is parsed the macro for the execution of the 'sendMedia' command is loaded, and initialized by the values passed in the script. Finally it is executed to make calls from Network Communication Broker to the Synthesis Engine.

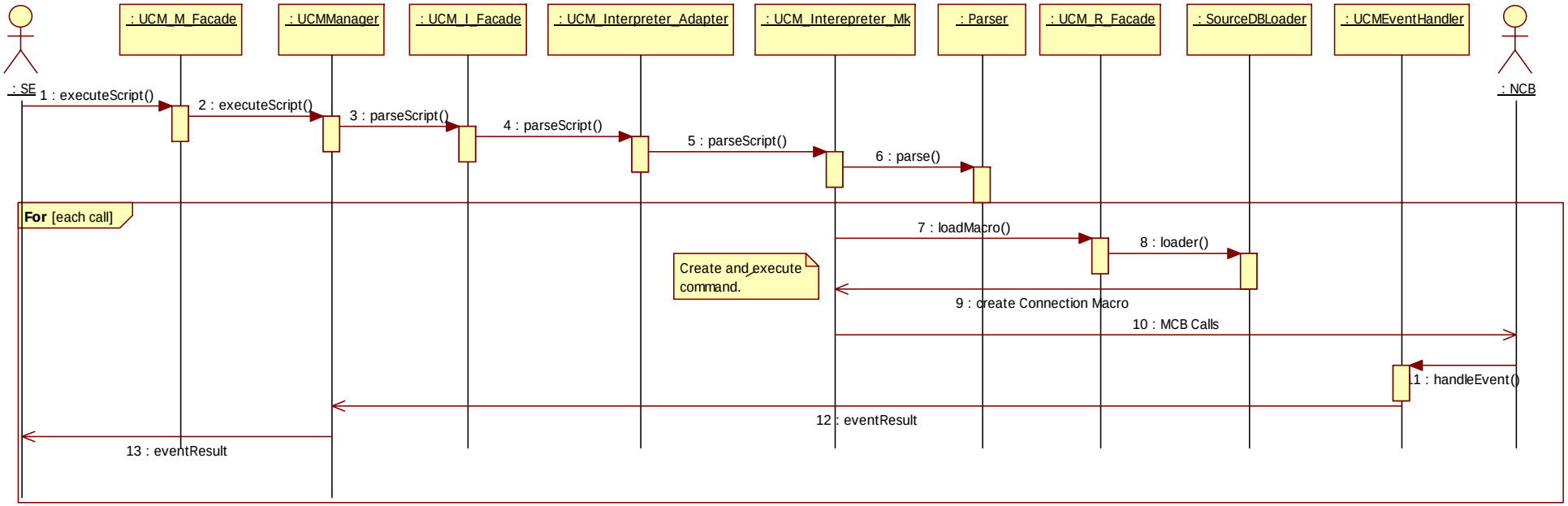


Fig. 6.3.8 Sequence Diagram – Create 2-Way Audio

Description: This sequence diagram represents the steps that UCM takes for handling a control script containing a set of commands from SE to create a complete 2-way audio communication. A control script is received from the Synthesis Engine, by the manager, then the manager delegates it to the interpreter which then parses it. Once the script is parsed the macro for the execution of each of the commands is loaded, and initialized by the values passed in the script. Finally each command is executed and calls are made from the Network Communication Broker to the Synthesis Engine.

6.4 Detailed Class Design

6.4.1 Design Patterns

Singleton: Design pattern used to restrict instantiation of a class to one object. This design patter was chosen for the Manager 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 Manager were floating around at all times.

Command: Design pattenr in which objects are used to represent actions. A command object encapsulates an action and its parameters. The command pattern was chosen to guarantee a control over the execution calls to the NCB API. This way we have control when and whether or not a action gets to execute, thus preventing or canceling undesirable behavior, or even threats as well as the logging of actions take to restart the system in the same state that it was in the event of a failure.

Facade: This design pattern was chosen to decouple the subsystems from each other. The facade pattern was also chosen to improve future evolution of the system.

Strategy: The strategy design pattern was chosen for the implementation for the repository subsystem classes. The use of this design pattern will allow for future distinct implementation of repositories. As of the macro data is stored in a database, with the use of the strategy pattern the implementation of different repositories is simplified.

6.4.2 Class Description

UCM_M_Façade (Control Object): Façade class that provides an interface into the UCM_Manager subsystem. The purpose of this class is to aid future expansion of this subsystem. Refer to diagram C.1 in Appendix C and class and Appendix E for class interface for classes and OCL statements.

UCMManager: Controls the workflow inside the UCM system. This class also notifies the overlying system of any event that are specific to it. Refer to diagram C.1 in Appendix C and class and Appendix E for class interface.

UCM_I_Façade (Control Object): Façade class that provides an interface into the UCM_Interpreter subsystem. The purpose of this class is to aid future expansion of this subsystem. Refer to diagram C.2 in Appendix C and class and Appendix E for class interface for classes and OCL statements.

UCM_Interepreter_Mk: This is the microkernel as in the architecture pattern. This class control the parsing of scripts, the loading of macros, and the queuing and execution of commands. Refer to diagram C.2 in Appendix C and class and Appendix E for class interface.

UCM_Interpreter_Adapter: Adapter class as in the microkernel architecture pattern. It makes the parse script requests to the UCM_Interepreter_Mk class. Refer to diagram C.2 in Appendix C and class and Appendix E for class interface.

Command (Interface): This is the Command Design Pattern interface. Refer to diagram C.2 in Appendix C and class and Appendix E for class interface.

MacroCommand: This class handles encapsulation of the execution of macros inside our system. This is part of the incorporation of the command design pattern into our system. Refer to diagram C.2 in Appendix C and class and Appendix E for class interface.

MacroNode: This class encapsulates the script editor and the parameters needed to execute a macro. This node is then encapsulated by the MacroCommand class. This is the 'Receiver' of the command design pattern. Refer to diagram C.2 in Appendix C and class and Appendix E for class interface.

Macro: This class encapsulates all the data specific to any macro. This object simplifies the transfer of information across subsystems dealing with macros. Refer to diagram C.2 in Appendix C and class and Appendix E for class interface.

Parser: This class parses the control scripts that are passed down from the Synthesis Engine and puts them in a data structure that is easier to handle and understand. Refer to diagram C.2 in Appendix C and class and Appendix E for class interface.

Script: Data structure that holds a control script in memory. Refer to diagram C.2 in Appendix C and class and Appendix E for class interface.

Call: Call holds function calls in a given control script. Refer to diagram C.2 in Appendix C and class and Appendix E for class interface.

MacroInterpreter: MacroInterpreter encapsulates the creation of a script evaluator object which will be executed using Janino inside the MacroCommand execute. The main purpose of this class is to receive a collection of parameter values and a macro object, which will contain parsed macro information. The MacroInterpreter will then encapsulate the data in a MacroNode for later execution. Refer to diagram C.2 in Appendix C and class and Appendix E for class interface.

UCM_R_Façade (Control Object): Façade class that provides an interface into the UCM_Repository subsystem. The purpose of this class is to aid future expansion of this subsystem. Refer to diagram C.3 in Appendix C and class and Appendix E for class interface for classes and OCL statements.

Sources: This class provides the interface to the Source Loader classes. Source Loaders can be either from the File System or from a Database. This class plays the role of AlgorithmInterface in the Strategy design pattern which helps to choose either of the strategies(FileSys or Database) at runtime. Refer to diagram C.3 in Appendix C and class and Appendix E for class interface.

MacroLoader: The MacroLoader class retrieves, from a given Source (a Database, a File System, etc), the information necessary and creates a Macro. Refer to diagram C.3 in Appendix C and class and Appendix E for class interface.

SourceDBLoader: This class retrieves a Macro from a database given a function name. There is no overloading of a function name. The schema for the table Macros, stored in the

Repository db, is {name:string, returnType:string, paramTypeList:string, paramNameList:string, script:string}. Refer to diagram C.3 in Appendix C and class and Appendix E for class interface.

SourceFileSysLoader: This class retrieves a Macro from a file system given a function name. There is no overloading of a function name. This has been created for future implementation availability. Refer to diagram C.3 in Appendix C and class and Appendix E for class interface.

UCMEventHandler (Control Object): This class handles all the possible events that the NCB layer fires. It then dispatches the events to the UCMManager for further processing, by calling the notifyEvent interface of UCMManager. The current events that are handled by the UCMEventHandler include: NotifyLoginReply_Event, SendSchemaReply_Event, PartyAddedReply_Event, ConnectionCreatedReply_Event, PartyRemovedReply_Event. Refer to diagram C.4 in Appendix C and class and Appendix E for class interface for classes and OCL statements.

ConnectionCreatedReply_Event: This class encapsulates the event source and the reply messages of invoking *createSession* coming from NCB into a single event. Refer to diagram C.4 in Appendix C and class and Appendix E for class interface.

NotifyLoginReply_Event: This class encapsulates the event source and the reply messages of invoking *login* method coming from NCB into a single event. Refer to diagram C.4 in Appendix C and class and Appendix E for class interface.

SendSchemaReply_Event: This class encapsulates the event source and the reply messages of invoking *sendSchema* coming from NCB into a single event. Refer to diagram C.4 in Appendix C and class and Appendix E for class interface.

PartyAddedReply_Event: This class encapsulates the event source and the reply messages of invoking *addParty* coming from NCB into a single event. Refer to diagram C.4 in Appendix C and class and Appendix E for class interface.

PartyRemovedReply_Event: This class encapsulates the event source and the reply messages of invoking *removeParty* coming from NCB into a single event. Refer to diagram C.4 in Appendix C and class and Appendix E for class interface.

UCMExceptionHandler (Control Object): This class handles the possible exceptions resulting in the runtime of the script interpretation process. Currently, the exception resolution includes printing out the context information of the exception, and in the future, it might include exception recovery or other complicated exception resolution mechanisms. The UCMExceptionHandler can now handle the following exceptions: MacroNotFoundException, IllegalArgumentException, PartyNotFoundException, NoSessionException, DataNotFoundException, InvalidScriptException. Refer to diagram C.5 in Appendix C and class and Appendix E for class interface for classes and OCL statements.

MacroNotFoundException: This class defines the exception thrown by the macro loader when the macro is not found in the local repository. Refer to diagram C.5 in Appendix C and class and Appendix E for class interface.

IllegalArgumentException: This class defines the exception thrown by the macro Interpreter when the macro has an unexpected argument passed to it. Refer to diagram C.5 in Appendix C and class and Appendix E for class interface.

PartyNotFoundException: This class defines the exception thrown by the UCM Interpreter when it is trying to remove a party that is not in the current connection. Refer to diagram C.5 in Appendix C and class and Appendix E for class interface.

NoSessionException: This class defines the exception thrown by the UCM Interpreter when it is trying to add participants or send data in the connection that has no corresponding session in the NCB layer. Refer to diagram C.5 in Appendix C and class and Appendix E for class interface.

DataNotFoundException: This class defines the exception thrown by the UCM Interpreter when it is trying to send data that could not be found locally or on the network. Refer to diagram C.5 in Appendix C and class and Appendix E for class interface.

InvalidScriptException: This class defines the exception thrown by the UCM Interpreter when it is trying to pass the string received from Synthesis Engine. Refer to diagram C.5 in Appendix C and class and Appendix E for class interface.

7. Testing Process

In this chapter the different testing methods will be introduced. An example of each will be presented, starting with the System test, in which the system is tested as a whole based on the Use Cases. The Subsystem Test will focus on one specific subsystem and make sure it behaves according to the specifications. Last, the Unit Tests for one specific class will be presented. They will ensure the correctness of the code.

7.1 System Test

The System tests described below were developed in an attempt to test different combinations of events and function calls that would ensure that all classes were called upon. The System Tests were developed to be executed by the UCM in the same manner that the macros are executed. The source code for the System Tests are stored in the Repository and is executed dynamically, exactly like a real UCM scenario.

Identifier:	UCM_T_1
Owner/Creator:	Raidel Batista
Version:	V1
Name:	Test Login
Requirement ID:	UCM_1_Login
Purpose:	The purpose of this test case is to test the Login Use Case
Dependencies:	None
Environment/ Configuration:	None
Initialization:	All three layers NCB, SE, and UCM must be initialized. SE and NCB must have been attached to the UCM_Manager.
Finalization:	If the log in fails the application must throw a LoginException which should be caught by the exception handler and triggered as an event by the EventHandler.
Actions:	Try to login a user to the NCB, retrieve the user's schema and create a user profile.
Input data:	Login control script containing the username = 'a' and password = 'password123'.
Expected results:	The application will notify the Synthesis Engine with a UserProfileCreatedEvent passing the profile of the user.

Identifier:	UCM_T_02
Owner/Creator:	Guangqiang Zhao
Version:	V1
Name:	Create Connection
Requirement ID:	UCM_03_CreateConnection
Purpose:	The purpose of this test case is to test the Create Connection Use Case
Dependencies:	UCM_T_01
Environment/ Configuration:	None
Initialization:	The different layers of the system, SE, UCM and NCB should start up and UCM is ready to take in and executes control scripts
Finalization:	If the connection is not created successfully, an exception should be thrown by the lower layer NCB.
Actions:	The system will pass down a control script requesting to log in and create connections
Input data:	The control scripts from SE contain the login request and the createConnections request, the login macro includes the user name and password and the createConnections macro includes connection id, as follows: "login (\"a\",\"password123\")\n" "createConnections (\"c1\")\n"
Expected results:	The user is successfully logged in and the connection should be created by the lower layer, indicated by events notifying the successful creation of the connection ConnectionCreatedReply_Event and UserProfileCreatedEvent

Identifier:	UCM_T_03
Owner/Creator:	Marylurdys Hernandez
Version:	V1
Name:	Test Add Participants with Settings
Requirement ID:	UCM_T_03
Purpose:	The purpose of this test case is to test the Add Participant Use Case using the creation of a new connection.

Dependencies:	Login and CreateConnection
Environment/ Configuration:	None
Initialization:	The user “a” is logged in and a connection “c1” is created.
Finalization:	If this fails, a UCM_Exception is thrown.
Actions:	Try to add a participant “b” after the user “a” is logged in and a connection “c1” has been created.
Input data:	Username “a”, password “password123”, connection id “c1” and the username of the participant to be added “b”.
Expected results:	A notification from NCB confirming the addition of a participant to the connection.

Identifier:	UCM_T_04
Owner/Creator:	Marylurdys Hernandez
Version:	V1
Name:	Test Add Participants Without Settings
Requirement ID:	UCM_T_04
Purpose:	The purpose of this test case is to test the Add Participant Use Case but using a non-existing connection.
Dependencies:	Login
Environment/ Configuration:	None
Initialization:	The user “a” is logged in.
Finalization:	None
Actions:	Try to add a participant “b” after the login of “a” is performed using a non-existing connection “c1”.
Input data:	Username “a”, password “password123”, connection id “c1” and the username of the participant to be added “b”.
Expected results:	A NoSessionException should be thrown notifying the non-existing connection.

Identifier:	UCM_T_5

Owner/Creator:	Raidel Batista
Version:	V1
Name:	Test send schema with settings
Requirement ID:	UCM_01, UCM_02, UCM_03
Purpose:	The purpose of this test case is to test the Send Schema (with settings) command
Dependencies:	UCM_T_01, UCM_T_02, UCM_T_03
Environment/ Configuration:	None
Initialization:	All three layers NCB, SE, and UCM must be initialized. SE and NCB must have been attached to the UCM_Manager.
Finalization:	If the connection fails a NoSessionException will be returned and trigger the appropriate event. If the data schema is unable to the sent by the ncb a ControlSchemaNotSentException will be triggered. Similarly if the data schema can not be sent a DataSchemaNotSentException will be triggered.
Actions:	The system will call the login event from SE and will be logged in The system has will have created a connection and a session IDs The system will have added participants to the connection such as there are at least two participants in the connection. The NCB will send the control and data schema.
Input data:	The Send Schema control script containing the connection id, the sender id, a list of receivers, the control schema and the data schema. "login (\"a\",\"password123\")\n"); "createConnections (\"c1\")\n" "addParticipants(\"c1\",\"b,c\")\n" "sendSchema(\"c1\",\"a\",\"b,c\",\"control_xcml\",\"data_xcml\")\n"
Expected results:	The application will notify the Synthesis Engine with a notifySendSchemaReply_Event passing the 'true' as the parameter.

Identifier:	UCM_T_6
Owner/Creator:	Raidel Batista
Version:	V1
Name:	Test send schema without settings
Requirement ID:	None
Purpose:	The purpose of this test case is to test the Send Schema (without settings)
Dependencies:	None
Environment/ Configuration:	None
Initialization:	All three layers NCB, SE, and UCM must be initialized. SE and NCB must have been attached to the UCM_Manager.

Finalization:	If the connection fails a NoSessionException will be returned and trigger the appropriate event. If the data schema is unable to the sent by the ncb a ControlSchemaNotSentException will be triggered. Similarly if the data schema can not be sent a DataSchemaNotSentException will be triggered.
Actions:	The NCB will send the control and data schema.
Input data:	The Send Schema control script containing the connection id, the sender id, a list of receievers, the control schema and the data schema. "sendSchema(\"c1\",\"a\",\"b,c\",\"control_xcml\",\"data_xcml\")\n"
Expected results:	SE is notified of a NoSessionException.

Identifier:	UCM_T_07
Owner/Creator:	Guangqiang Zhao
Version:	V1
Name:	Enable Media Initiation with settings
Requirement ID:	UCM_24_EnableMediaInitiator
Purpose:	The purpose of this test case is to test the Enable Media Initiator use case with settings
Dependencies:	None
Environment/ Configuration:	None
Initialization:	The system starts up and ready to take in and executes control scripts
Finalization:	If problems occur within the intermediate stage, informative error messages should be shown.
Actions:	Since this is a behavior of an internal system component, we do not consider the user actions.
Input data:	The sequence of control scripts from the Synthesis Engine including the login, createConnection and sendSchema, • login macro contains the user name and password • createConnection macro includes connection id • addParticipants macro includes connection id and list of participants • sendSchema macro includes connection id, sender id, list of participants, control schema and data schema • enableMediaInitiator macro includes connection id and media name "login (\"a\",\"password123\")\n"); "createConnections (\"c1\")\n" "addParticipants(\"c1\",\"b,c\")\n" "sendSchema(\"c1\",\"a\",\"b,c\",\"control_xcml\",\"data_xcml\")\n" "enableMediaInitiator(\"c1\",\"audio\")"

Expected results:	The media should be initiated from scratch, all the way from logged in, create connection, add participants, send schema to enable media initiator. Event: ● UserProfileCreatedEvent ● ConnectionCreatedReply_Event ● PartyAddedReply_Event ● SendSchemaReply_Event

Identifier:	UCM_T_08
Owner/Creator:	Guangqiang Zhao
Version:	V1
Name:	Test Enable Media Initiation without settings
Requirement ID:	UCM_24_EnableMediaInitiator
Purpose:	The purpose of this test case is to test that an exception could be handled and notified during the Enable Media Initiation use case
Dependencies:	UCM_T_02, UCM_T_03, UCM_T_05
Environment/ Configuration:	None
Initialization:	The different layers of the system, SE, UCM and NCB should start up and UCM is ready to take in and executes control scripts
Finalization:	If the media is not initialized successfully due to lack of connection, NoSessionException should be returned.
Actions:	The SE will pass down a control script requesting to enable media initiator. NCB will start to send the media stream to remote participants.
Input data:	The enable Media Initiator control script contains the connection id, and the name of the media, as follows: “enableMediaInitiator(\"c1\",\"audio\")
Expected results:	An NoSessionException should be thrown

Identifier:	UCM_T_09
Owner/Creator:	Eduardo Monteiro
Version:	V1
Name:	Decline Connection with settings.
Requirement ID:	UCM_T_14
Purpose:	The purpose of this test case is to make sure Decline Connection works as expected.
Dependencies:	None
Environment/ Configuration:	None
Initialization:	The user logs in with username A, password P, creates a connection C1, adds two participants, B and C, initiates audio.
Actions:	Decline the connection C.
Input data:	A = “a” P = “password123” C1 = “c1” B = “b” C = “c”
Expected results:	The connection is declined.

Identifier:	UCM_T_10
Owner/Creator:	Abhishek Bhattacharya
Version:	V1
Name:	Test to remove existing participant with settings
Requirement ID:	
Purpose:	The purpose of this test case is to test the removeParticipants event when an active connection is present with the participant to be removed is already added as an participant in the connection. This call may be used by any active participant in the connection to remove any participant also in the connection.
Dependencies:	None
Environment/ Configuration:	The SE and NCB engines are reset at the beginning of the test case so they are ready for the new settings.
Initialization:	Users must be logged in and there must be an active connection present with some active participants in the connection. The schema negotiation is

	completed between the initiator and the receivers and a particular media (audio or video) is also enabled in the connection.
Finalization:	If the removal of existing participants fail then it should throw an exception.
Actions:	Try to remove an existing participant in the connection.
Input data:	1. Login command invoked with parameters : username='A' and password='password123'. 2. CreateConnection command invoked with parameters : connectionID='c1' 3. AddParticipants command invoked with parameters : connectionID='c1' and participantID='b' and 'c'. 4. SendSchema command invoked with parameters : connectionID='c1', initiatorID='a', participantID='b' and 'c', ControlSchema='control_xcml', DataSchema='data_schema'. 5. EnableMediaInitiator command invoked with parameters : connectionID='c1', mediaType='audio' 6. RemoveParticipant command invoked with parameters : connectionID='c1', participantID='b'.
Expected results:	1. NotifyLoginReply event is captured after successful login execution in UCM. 2. ConnectionCreatedReply event is captured after successful creation of connection in UCM. 3. PartyAddedReply event is captured after successful addition of participants into the connection from UCM. 4. SendSchemaReply event is captured after successful operation of sending control schema to the participants from UCM. 5. MediaInitiatorEnableReply event is captured after successful operation of enabling media from UCM. 6. PartyRemovedReply event is captured after successful removal of participant in the connection from UCM. 7. UserProfileCreated event is captured after successful creation of user profile in SE.

Identifier:	UCM_T_11
Owner/Creator:	Abhishek Bhattacharya
Version:	V1
Name:	Test to remove non-existing participant with settings
Requirement ID:	
Purpose:	The purpose of this test case is to test the exception condition to be thrown when a non-existing participant is being tried to be removed from the connection i.e. a removeParticipant event is called on a non-existing participant.

Dependencies:	None
Environment/ Configuration:	The SE and NCB engines are reset at the beginning of the test case so they are ready for the new settings.
Initialization:	Users must be logged in and there must be an active connection present with some active participants in the connection. The schema negotiation is completed between the initiator and the receivers and a particular media (audio or video) is also enabled in the connection.
Finalization:	If the removal of the non-existing participant fails then it should throw an exception.
Actions:	Try to remove a non-existing participant from the connection.
Input data:	1. Login command invoked with parameters : username='A' and password='password123'. 2. CreateConnection command invoked with parameters : connectionID='c1' 3. AddParticipants command invoked with parameters : connectionID='c1' and participantID='b' and 'c'. 4. SendSchema command invoked with parameters : connectionID='c1', initiatorID='a', participantID='b' and 'c', ControlSchema='control_xcml', DataSchema='data_xcml'. 5. EnableMediaInitiator command invoked with parameters : connectionID='c1', mediaType='audio' 6. RemoveParticipant command invoked with parameters : connectionID='c1', participantID='d'.
Expected results:	1. NotifyLoginReply event is captured after successful login execution in UCM. 2. ConnectionCreatedReply event is captured after successful creation of connection in UCM. 3. PartyAddedReply event is captured after successful addition of participants into the connection from UCM. 4. SendSchemaReply event is captured after successful operation of sending control schema to the participants from UCM. 5. MediaInitiatorEnableReply event is captured after successful operation of enabling media from UCM. 6. PartyNotFound exception is thrown after unsuccessful removal of participant in the connection from UCM. 7. UserProfileCreated event is captured after successful creation of user profile in SE.

Identifier:	UCM_T_13
Owner/Creator:	Eduardo Monteiro
Version:	V1
Name:	Logout with failed remove party.

Purpose:	The purpose of this test case is to make sure logout works when removing a party fails.
Dependencies:	None
Environment/ Configuration:	None
Initialization:	The user logs in with username A, password P, creates a connection C1, adds two participants, B and C, initiates audio.
Actions:	Remove a non-existing participant and logout.
Input data:	A = “a” P = “password123” C1 = “c1” B = “b” C = “c”
Expected results:	A UserNotFound Exception is thrown. The system logs out successfully.

Identifier:	UCM_T_14
Owner/Creator:	Eduardo Monteiro
Version:	V1
Name:	Logout with remove party.
Purpose:	The purpose of this test case is to make sure logout works when removing a party.
Dependencies:	None
Environment/ Configuration:	None
Initialization:	The user logs in with username A, password P, creates a connection C1, adds two participants, B and C, initiates audio.
Actions:	Remove participants and logout.
Input data:	A = “a” P = “password123” C1 = “c1” B = “b” C = “c”
Expected results:	The system logs out successfully.

Identifier:	UCM_T_15
Owner/Creator:	Raidel Batista
Version:	V1
Name:	Test Login
Requirement ID:	UCM_1_Login
Purpose:	The purpose of this test case is to test the Login Use Case
Dependencies:	None
Environment/ Configuration:	None
Initialization:	All three layers NCB, SE, and UCM must be initialized. SE and NCB must have been attached to the UCM_Manager.
Finalization:	If the log in fails the application must throw a LoginException which should be caught by the exception handler and triggered as an event by the EventHandler.
Actions:	Try to login a user to the NCB, retrieve the user's schema and create a user profile.
Input data:	Login control script containing the username = 'frank hernandez' and password = 'password567'.
Expected results:	The application will notify the Synthesis Engine with a UserProfileCreatedEvent passing the profile of the user.

Identifier:	UCM_T_16
Owner/Creator:	Guangqiang Zhao
Version:	V1
Name:	Create Connection
Requirement ID:	UCM_03_CreateConnection
Purpose:	The purpose of this test case is to test the Create Connection Use Case
Dependencies:	UCM_T_15
Environment/ Configuration:	None
Initialization:	The different layers of the system, SE, UCM and NCB should start up and UCM is ready to take in and executes control scripts
Finalization:	If the connection is not created successfully, an exception should be thrown by the lower layer NCB.
Actions:	The system will pass down a control script requesting to log in and create connections
Input data:	The control scripts from SE contain the login request and the createConnections request, the login macro includes the user name and password and the createConnections macro includes connection id, as

	follows: "login (\"frank hernandez\",\"password567\")\n" "createConnections (\"d1\")\n"
Expected results:	The user is successfully logged in and the connection should be created by the lower layer, indicated by events notifying the successful creation of the connection ConnectionCreatedReply_Event and UserProfileCreatedEvent

Identifier:	UCM_T_17
Owner/Creator:	Marylurdys Hernandez
Version:	V1
Name:	Test Add Participants with Settings
Requirement ID:	UCM_16
Purpose:	The purpose of this test case is to test the Add Participant Use Case using the creation of a new connection.
Dependencies:	Login and CreateConnection
Environment/ Configuration:	None
Initialization:	The user “frank hernandez” is logged in and a connection “d1” is created.
Finalization:	If this fails, a UCM_Exception is thrown.
Actions:	Try to add a participant “b” after the user “frank hernandez” is logged in and a connection “d1” has been created.
Input data:	Username “frank hernandez”, password “password567”, connection id “d1” and the username of the participant to be added “x”.
Expected results:	A notification from NCB confirming the addition of a participant to the connection.

Identifier:	UCM_T_18
Owner/Creator:	Marylurdys Hernandez
Version:	V1
Name:	Test Add Participants Without Settings
Requirement ID:	UCM_04

Purpose:	The purpose of this test case is to test the Add Participant Use Case but using a non-existing connection.
Dependencies:	Login
Environment/ Configuration:	None
Initialization:	The user “frank hernandez” is logged in.
Finalization:	None
Actions:	Try to add a participant “x” after the login of “frank hernandez” is performed using a non-existing connection “d1”.
Input data:	Username “frank hernandez”, password “password567”, connection id “d1” and the username of the participant to be added “x”.
Expected results:	A NoSessionException should be thrown notifying the non-existing connection.

Identifier:	UCM_T_19
Owner/Creator:	Raidel Batista
Version:	V1
Name:	Test send schema with settings
Requirement ID:	UCM_01, UCM_02, UCM_03
Purpose:	The purpose of this test case is to test the Send Schema (with settings) command
Dependencies:	UCM_T_15, UCM_T_16, UCM_T_17
Environment/ Configuration:	None
Initialization:	All three layers NCB, SE, and UCM must be initialized. SE and NCB must have been attached to the UCM_Manager.
Finalization:	If the connection fails a NoSessionException will be returned and trigger the appropriate event. If the data schema is unable to the sent by the ncb a ControlSchemaNotSentException will be triggered. Similarly if the data schema can not be sent a DataSchemaNotSentException will be triggered.
Actions:	The system will call the login event from SE and will be logged in The system has will have created a connection and a session IDs The system will have added participants to the connection such as there are at least two participants in the connection. The NCB will send the control and data schema.

Input data:	The Send Schema control script containing the connection id, the sender id, a list of receievers, the control schema and the data schema. "login (\"frank hernandez\",\"password567\")\n"); "createConnections (\"d1\")\n" "addParticipants(\"d1\",\"x,y\")\n" "sendSchema(\"c1\",\"frank hernandez\",\"x,y\",\"control_xcml\",\"data_xcml\")\n"
Expected results:	The application will notify the Synthesis Engine with a notifySendSchemaReply_Event passing the 'true' as the parameter.

Identifier:	UCM_T_20
Owner/Creator:	Raidel Batista
Version:	V1
Name:	Test send schema without settings
Requirement ID:	None
Purpose:	The purpose of this test case is to test the Send Schema (without settings)
Dependencies:	None
Environment/ Configuration:	None
Initialization:	All three layers NCB, SE, and UCM must be initialized. SE and NCB must have been attached to the UCM_Manager.
Finalization:	If the connection fails a NoSessionException will be returned and trigger the appropriate event. If the data schema is unable to the sent by the ncb a ControlSchemaNotSentException will be triggered. Similarly if the data schema can not be sent a DataSchemaNotSentException will be triggered.
Actions:	The NCB will send the control and data schema.
Input data:	The Send Schema control script containing the connection id, the sender id, a list of receievers, the control schema and the data schema. "sendSchema(\"d1\",\"frank hernandez\",\"x,y\",\"control_xcml\",\"data_xcml\")\n"
Expected results:	SE is notified of a NoSessionException.

Identifier:	UCM_T_21
Owner/Creator:	Guangqiang Zhao
Version:	V1
Name:	Enable Media Initiation with settings
Requirement ID:	UCM_24_EnableMediaInitiator

Purpose:	The purpose of this test case is to test the Enable Media Initiator use case with settings
Dependencies:	None
Environment/ Configuration:	None
Initialization:	The system starts up and ready to take in and executes control scripts
Finalization:	If problems occur within the intermediate stage, informative error messages should be shown.
Actions:	Since this is a behavior of an internal system component, we do not consider the user actions.
Input data:	The sequence of control scripts from the Synthesis Engine including the login, createConnection and sendSchema, ● login macro contains the user name and password ● createConnection macro includes connection id ● addParticipants macro includes connection id and list of participants ● sendSchema macro includes connection id, sender id, list of participants, control schema and data schema ● enableMediaInitiator macro includes connection id and media name "login (\"frank hernandez\",\"password567\")\n"); "createConnections (\"d1\")\n" "addParticipants(\"d1\",\"x,y\")\n" "sendSchema(\"d1\",\"frank hernandez\",\"x,y\",\"control_xcml\",\"data_xcml\")\n" "enableMediaInitiator(\"d1\",\"audio\")"
Expected results:	The media should be initiated from scratch, all the way from logged in, create connection, add participants, send schema to enable media initiator. Event: ● UserProfileCreatedEvent ● ConnectionCreatedReply_Event ● PartyAddedReply_Event ● SendSchemaReply_Event

Identifier:	UCM_T_22
Owner/Creator:	Guangqiang Zhao
Version:	V1
Name:	Test Enable Media Initiation without settings
Requirement ID:	UCM_24_EnableMediaInitiator
Purpose:	The purpose of this test case is to test that an exception could be handled and notified during the Enable Media Initiation use case
Dependencies:	UCM_T_16, UCM_T_17, UCM_T_19

Environment/ Configuration:	None
Initialization:	The different layers of the system, SE, UCM and NCB should start up and UCM is ready to take in and executes control scripts
Finalization:	If the media is not initialized successfully due to lack of connection, NoSessionException should be returned.
Actions:	The SE will pass down a control script requesting to enable media initiator. NCB will start to send the media stream to remote participants.
Input data:	The enable Media Initiator control script contains the connection id, and the name of the media, as follows: “enableMediaInitiator(\"d1\",\"audio\")
Expected results:	An NoSessionException should be thrown

Identifier:	UCM_T_23
Owner/Creator:	Abhishek Bhattacharya
Version:	V1
Name:	Test to remove existing participant with settings
Requirement ID:	
Purpose:	The purpose of this test case is to test the removeParticipants event when an active connection is present with the participant to be removed is already added as an participant in the connection. This call may be used by any active participant in the connection to remove any participant also in the connection.
Dependencies:	None
Environment/ Configuration:	The SE and NCB engines are reset at the beginning of the test case so they are ready for the new settings.
Initialization:	Users must be logged in and there must be an active connection present with some active participants in the connection. The schema negotiation is completed between the initiator and the receivers and a particular media (audio or video) is also enabled in the connection.
Finalization:	If the removal of existing participants fail then it should throw an exception.
Actions:	Try to remove an existing participant in the connection.
Input data:	1. Login command invoked with parameters : username=’frank hernandez’ and password=’password567’.

	2. CreateConnection command invoked with parameters : connectionID=’d1’ 3. AddParticipants command invoked with parameters : connectionID=’d1’ and participantID=’x’ and ‘y’. 4. SendSchema command invoked with parameters : connectionID=’d1’, initiatorID=’a’, participantID=’x’ and ‘y’, ControlSchema=’control_xcml’, DataSchema=’data_schema’. 5. EnableMediaInitiator command invoked with parameters : connectionID=’d1’, mediaType=’audio’ 6. RemoveParticipant command invoked with parameters : connectionID=’d1’, participantID=’x’.
Expected results:	1. NotifyLoginReply event is captured after successful login execution in UCM. 2. ConnectionCreatedReply event is captured after successful creation of connection in UCM. 3. PartyAddedReply event is captured after successful addition of participants into the connection from UCM. 4. SendSchemaReply event is captured after successful operation of sending control schema to the participants from UCM. 5. MediaInitiatorEnableReply event is captured after successful operation of enabling media from UCM. 6. PartyRemovedReply event is captured after successful removal of participant in the connection from UCM. 7. UserProfileCreated event is captured after successful creation of user profile in SE.

Identifier:	UCM_T_24
Owner/Creator:	Abhishek Bhattacharya
Version:	V1
Name:	Test to remove non-existing participant with settings
Requirement ID:	
Purpose:	The purpose of this test case is to test the exception condition to be thrown when a non-existing participant is being tried to be removed from the connection i.e. a removeParticipant event is called on a non-existing participant.
Dependencies:	None
Environment/ Configuration:	The SE and NCB engines are reset at the beginning of the test case so they are ready for the new settings.
Initialization:	Users must be logged in and there must be an active connection present with some active participants in the connection. The schema negotiation is completed between the initiator and the receivers and a particular media (audio or video) is also enabled in the connection.

Finalization:	If the removal of the non-existing participant fails then it should throw an exception.
Actions:	Try to remove a non-existing participant from the connection.
Input data:	1. Login command invoked with parameters : username=’frank hernandez’ and password=’password567’. 2. CreateConnection command invoked with parameters : connectionID=’d1’ 3. AddParticipants command invoked with parameters : connectionID=’d1’ and participantID=’x’ and ‘y’. 4. SendSchema command invoked with parameters : connectionID=’d1’, initiatorID=’a’, participantID=’x’ and ‘y’, ControlSchema=’control_xcml’, DataSchema=’data_xcml’. 5. EnableMediaInitiator command invoked with parameters : connectionID=’d1’, mediaType=’audio’ 6. RemoveParticipant command invoked with parameters : connectionID=’d1’, participantID=’z’.
Expected results:	1. NotifyLoginReply event is captured after successful login execution in UCM. 2. ConnectionCreatedReply event is captured after successful creation of connection in UCM. 3. PartyAddedReply event is captured after successful addition of participants into the connection from UCM. 4. SendSchemaReply event is captured after successful operation of sending control schema to the participants from UCM. 5. MediaInitiatorEnableReply event is captured after successful operation of enabling media from UCM. 6. PartyNotFound exception is thrown after unsuccessful removal of participant in the connection from UCM. 7. UserProfileCreated event is captured after successful creation of user profile in SE.

7.2 Subsystem Test

The Subsystem Test presented below tests the Repository Subsystem. This test ensures that all classes in the Repository package are called upon. It starts by initializing the repository and making sure that different database and file system types can be used. It then tests the insertion and retrieval of data into the repository. Error handling is also tested and misuse is ensured to be harmless to the system.

Identifier:	UCM_T_12
Owner/Creator:	Marylurdys Hernandez
Version:	V1

Name:	Test Repository Subsystem Interface
Requirement ID:	UCM_T_12
Purpose:	The purpose of this test case is to test the interface provided by the Repository Subsystem in terms of returning the correct Macro object that other subsystem request.
Dependencies:	None
Environment/ Configuration:	None
Initialization:	Create the UCM_R_Facade
Finalization:	If this fails, one or more of these exceptions are thrown: SourceDBLoader, MacroLoader, or UCM_R_Facade.
Actions:	Request to the UCM_R_Facade to load the Macro object for the "login" command
Input data:	The name of the macro to be loaded, "login"
Expected results:	Each parameter stored in the repository for the macro "login"

7.3 Unit Test

The Unit Tests were developed for the Script class. The correctness of the code is tested by creating a sample Script, adding a number of calls with different return types and parameter types and names, and making sure that they are stored correctly and can be retrieved according to the specifications. If all tests are successful, the Unit Test is also successful.

Identifier:	UCM_T_25
Owner/Creator:	Frank Hernandez
Version:	V1
Name:	Unit Test – UCM_M_Façade
Requirement ID:	UCM_1_Login
Purpose:	The purpose of this test case is to test the Login Use Case
Dependencies:	None
Environment/ Configuration:	None
Initialization:	All three layers NCB, SE, and UCM must be initialized. SE and NCB must have been attached to the UCM_Manager.

Finalization:	If the log in fails the application must throw a LoginException which should be caught by the exception handler and triggered as an event by the EventHandler.
Actions:	Try to login a user to the NCB, retrieve the user's schema and create a user profile.
Input data:	Login control script containing the username = 'a' and password = '*****'.
Expected results:	The application will notify the Synthesis Engine with a UserProfileCreatedEvent passing the profile of the user.

Identifier:	UCM_T_12
Owner/Creator:	Marylurdys Hernandez
Version:	V1
Name:	Unit Test - UCM_R_Facade
Requirement ID:	None
Purpose:	The purpose of this test case is to test the interface provided by the Repository Subsystem in terms of returning the correct Macro object that other subsystem request.
Dependencies:	None
Environment/ Configuration:	None
Initialization:	Create the UCM_R_Facade
Finalization:	If this fails, one or more of these exceptions are thrown: SourceDBLoader, MacroLoader, or UCM_R_Facade.
Actions:	Request to the UCM_R_Facade to load the Macro object for the "createConnection" command
Input data:	The name of the macro to be loaded, "createConnection"
Expected results:	Each parameter stored in the repository for the macro "createConnection"

7.4 Evaluation of Tests

While it is impossible to test for all combinations of user actions and control scripts, the tests tried to cover the most common subset of both. The tests try to cover the use cases that were

deemed critical and more frequent. In the end, testing the classes that are called by these scenarios and use cases will ensure that the coverage is at least satisfactory. Obscure combinations of control scripts and user actions might not have been tested, but due to time and scope of this project, the system was sufficiently tested.

Identifier:	UCM_T_1
Owner/Creator:	Raidel Batista
Version:	V1
Name:	Test Login
Actual results:	The application will notify the Synthesis Engine with a UserProfileCreatedEvent passing the profile of the user.
PASS/FAIL:	PASS.

Identifier:	UCM_T_02
Owner/Creator:	Guangqiang Zhao
Version:	V1
Name:	Create Connection
Actual result:	The system notifies the Synthesis Engine about the successful login and creation of connection via two events: ConnectionCreatedReply_Event and UserProfileCreatedEvent
PASS/FAIL:	PASS.

Identifier:	UCM_T_03
Owner/Creator:	Marylurdys Hernandez
Version:	V1
Name:	Test Add Participants with Settings
Actual results:	"A PartyAddedReply_Event was detected by UCM_Manager!"
PASS/FAIL:	PASS.

Identifier:	UCM_T_04
Owner/Creator:	Marylurdys Hernandez
Version:	V1

Name:	Test Add Participants Without Settings
Actual results:	"SE Received NoSessionException notification for sID: null". The session ID (sID) is null since the connection "c1" does not exist.
PASS/FAIL:	PASS.

Identifier:	UCM_T_5
Owner/Creator:	Raidel Batista
Version:	V1
Name:	Test send schema with settings
Actual Result	The application will notify the Synthesis Engine with a notifySendSchemaReply_Event passing the 'true' as the parameter
PASS/FAIL:	PASS.

Identifier:	UCM_T_6
Owner/Creator:	Raidel Batista
Version:	V1
Name:	Test send schema without settings
Actual results:	SE is notified of a NoSessionException.
PASS/FAIL:	PASS.

Identifier:	UCM_T_07
Owner/Creator:	Guangqiang Zhao
Version:	V1
Name:	Enable Media Initiation with settings
Actual result:	The system notified Synthesis Engine about the successful login, creation of connection, adding participants, sent schema and enabled media via the following events: • UserProfileCreatedEvent • ConnectionCreatedReply_Event • PartyAddedReply_Event • SendSchemaReply_Event • MediaInitiatorEnableReply_Event
PASS/FAIL:	PASS.

Identifier:	UCM_T_08
Owner/Creator:	Guangqiang Zhao
Version:	V1
Name:	Test Enable Media Initiation without settings

Actual result:	An UCM_Exception is thrown: NoSessionException
PASS/FAIL:	PASS.

Identifier:	UCM_T_09
Owner/Creator:	Eduardo Monteiro
Version:	V1
Name:	Decline Connection with settings.
Actual results:	The connection is declined.
PASS/FAIL:	PASS.

Identifier:	UCM_T_10
Owner/Creator:	Abhishek Bhattacharya
Version:	V1
Name:	Test to remove existing participant with settings
Actual result:	1. NotifyLoginReply_Event successful. 2. ConnectionCreatedReply_Event successful. 3. PartyAddedReply _Event successful. 4. SendSchemaReply_Event successful. 5. MediaInitiatorEnableReply_Event successful. 6. PartyRemovedReply_Event successful. 7. UserProfileCreatedEvent successful.
PASS/FAIL:	PASS.

Identifier:	UCM_T_11
Owner/Creator:	Abhishek Bhattacharya
Version:	V1
Name:	Test to remove non-existing participant with settings
Actual result:	1. NotifyLoginReply_Event successful. 2. ConnectionCreatedReply_Event successful. 3. PartyAddedReply _Event successful. 4. SendSchemaReply_Event successful. 5. MediaInitiatorEnableReply_Event successful. 6. PartyNotFoundException successful. 7. UserProfileCreatedEvent successful.
PASS/FAIL:	PASS.

Identifier:	UCM_T_12
Owner/Creator:	Marylurdys Hernandez
Version:	V1
Name:	Test Repository Subsystem
Actual results:	Success! The information stored in the repository for the "login" macro is returned, including name, parameters type list, parameters name list, the thrown exceptions list and the macro script.
PASS/FAIL:	PASS.

Identifier:	UCM_T_13
Owner/Creator:	Eduardo Monteiro
Version:	V1
Name:	Logout with failed remove party.
Actual results:	A UserNotFound Exception is thrown. The system logs out successfully.
PASS/FAIL:	PASS.

Identifier:	UCM_T_14
Owner/Creator:	Eduardo Monteiro
Version:	V1
Name:	Logout with remove party.
Actual results:	The system logs out successfully.
PASS/FAIL:	PASS.

Identifier:	UCM_T_15
Owner/Creator:	Raidel Batista
Version:	V1
Name:	Test Login
Actual results:	The application will notify the Synthesis Engine with a UserProfileCreatedEvent passing the profile of the user.
PASS/FAIL:	PASS.

Identifier:	UCM_T_16
Owner/Creator:	Guangqiang Zhao
Version:	V1
Name:	Create Connection

Actual result:	The system notifies the Synthesis Engine about the successful login and creation of connection via two events: ConnectionCreatedReply_Event and UserProfileCreatedEvent
PASS/FAIL:	PASS.

Identifier:	UCM_T_17
Owner/Creator:	Marylurdys Hernandez
Version:	V1
Name:	Test Add Participants with Settings
Actual results:	"A PartyAddedReply_Event was detected by UCM_Manager!"
PASS/FAIL:	PASS.

Identifier:	UCM_T_18
Owner/Creator:	Marylurdys Hernandez
Version:	V1
Name:	Test Add Participants Without Settings
Actual results:	"SE Received NoSessionException notification for sID: null". The session ID (sID) is null since the connection "d1" does not exist.
PASS/FAIL:	PASS.

Identifier:	UCM_T_19
Owner/Creator:	Raidel Batista
Version:	V1
Name:	Test send schema with settings
Actual Result	The application will notify the Synthesis Engine with a notifySendSchemaReply_Event passing the 'true' as the parameter
PASS/FAIL:	PASS.

Identifier:	UCM_T_20
Owner/Creator:	Raidel Batista
Version:	V1
Name:	Test send schema without settings
Actual results:	SE is notified of a NoSessionException.
PASS/FAIL:	PASS.

Identifier:	UCM_T_21
Owner/Creator:	Guangqiang Zhao
Version:	V1
Name:	Enable Media Initiation with settings
Actual result:	The system notified Synthesis Engine about the successful login, creation of connection, adding participants, sent schema and enabled media via the following events: ● UserProfileCreatedEvent ● ConnectionCreatedReply_Event ● PartyAddedReply_Event ● SendSchemaReply_Event ● MediaInitiatorEnableReply_Event
PASS/FAIL:	PASS.

Identifier:	UCM_T_22
Owner/Creator:	Guangqiang Zhao
Version:	V1
Name:	Test Enable Media Initiation without settings
Actual result:	An UCM_Exception is thrown: NoSessionException
PASS/FAIL:	PASS.

Identifier:	UCM_T_23
Owner/Creator:	Abhishek Bhattacharya
Version:	V1
Name:	Test to remove existing participant with settings
Actual result:	1. NotifyLoginReply_Event successful. 2. ConnectionCreatedReply_Event successful. 3. PartyAddedReply _Event successful. 4. SendSchemaReply_Event successful. 5. MediaInitiatorEnableReply_Event successful. 6. PartyRemovedReply_Event successful. 7. UserProfileCreatedEvent successful.
PASS/FAIL:	PASS.

Identifier:	UCM_T_24
Owner/Creator:	Abhishek Bhattacharya
Version:	V1
Name:	Test to remove non-existing participant with settings
Actual result:	1. NotifyLoginReply_Event successful. 2. ConnectionCreatedReply_Event successful. 3. PartyAddedReply _Event successful. 4. SendSchemaReply_Event successful. 5. MediaInitiatorEnableReply_Event successful. 6. PartyNotFoundException successful. 7. UserProfileCreatedEvent successful.
PASS/FAIL:	PASS.

Identifier:	UCM_T_25
Owner/Creator:	Frank Hernandez
Version:	V1
Name:	Unit Test – UCM_M_Façade

Actual results:	`EXECUTING MACRO: login .`
	`NCB Login called with userName:"a" and Password:"*****".`
	`NCB retrieve schema called with userName:"a" and Password:"*****".`
	`NCB createUserProfile called.`
	`Not a UCM_Event Redirecting to SE!`
	`UserProfileCreatedEvent is detected by SynthesisEngine!`
	`UserProfile Created!`

PASS/FAIL:	PASS.

Identifier:	UCM_T_26
Owner/Creator:	Marylurdys Hernandez
Version:	V1
Name:	Unit Test - UCM_R_Facade

Actual results:	`Macro: createConnections`
	`Return Type:`
	`cvm.ucm.handlers.exception.NoSessionException`
	`Parameters Type List: [java.lang.String]`
	`Parameters Name List: [connectionID]`
	`Script: import static java.lang.String; import static cvm.ucm.handlers.exception.NoSessionException;`
	`NoSessionException exception = null; String sID =`

	"s1"; try{ ncb.createSession(sID);
	if(!ncb.isCreatedSession()){ exception = new
	NoSessionException(sID);throw exception;}
	ncb.mapConnToSession(connectionID, sID);
	}catch(NoSessionException e){ return e;}
	ucmNotifier.notifyConnectionCreatedReply_Event(true);
	return exception;
	Exception List:
	[cvm.ucm.handlers.exception.NoSessionException]

PASS/FAIL:	PASS.

8. Glossary

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

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

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

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

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

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

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

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

UCM – Name of this system – User-Centric Communication Middleware.

NCB – Network Communication Broker.

SE – Name of the overlaying layer that sits atop UCM, Synthesis Engine.

CVM – Communication Virtual Machine.

RQ – Requirements.

PD – Product Design.

DD – Detailed Design.

CT – Cost & Unit Test.

IT – Integration Test.

9. Signature Page

----------	----------
Batista, Raidel	**Bhattacharya, Abhishek**
----------	----------
Hernandez, Frank	**Hernandez, Marylurdys**
----------	----------
Monteiro, Eduardo	**Zhao, Guangqiang**

10. Appendix

10.1 Appendix A – Project Schedule.

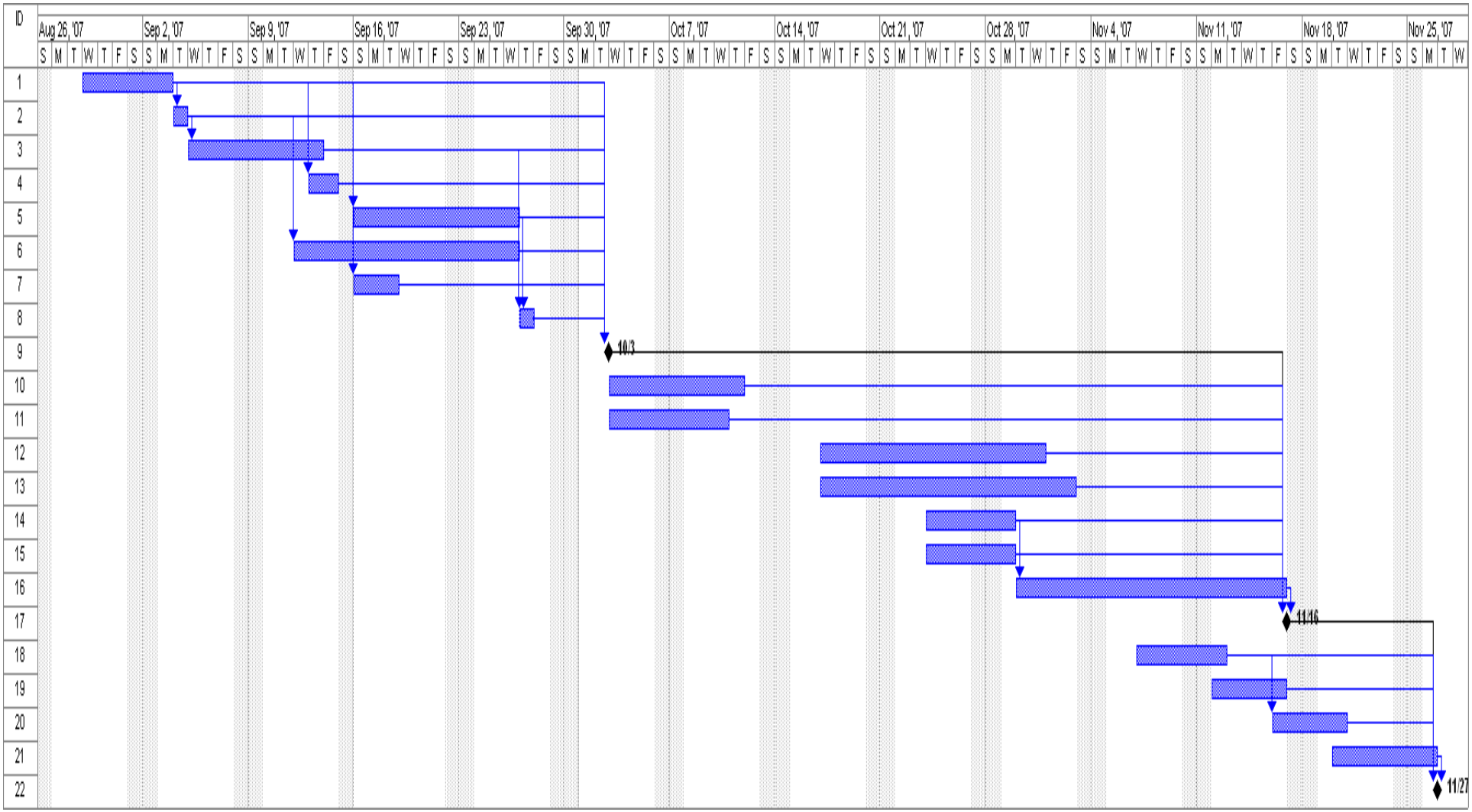


Fig A.2 Large Project Schedule

Description: This is the project schedule for the entire project. This project will cover from start to end of the semester.

10.2 Appendix B – Use Case with Nonfunctional Requirements

Use Case Login

***Use Case ID:** *UCM_01 – Login.*

Use Case Level: *Functional sub-use case.*

***Scenario:** A control script is received from the Synthesis Engine with the control command Login("a", "xxxxxxx").

- **Actor:** *Synthesis Engine (SE), Network Communication Broker (NCB).*
- **Pre-conditions:**

1. *There is an active Internet connection.*

- **Description:**

  Trigger: *Use case begins when the Synthesis Engine requests a login through the control script.*

  The system responds by:

  1. *UCM obtains the username and password from the control script.*
  2. *UCM calls the Login function passing the username and password as parameters.*
  3. *UCM receives an answer from the NCB stating if the Login was successful or not.*

- **Relevant requirements:** *none.*
- **Post-conditions:** *The user is logged in and ready to create connections.*

***Alternative Courses of Action** *No alternate action.*

Extensions: *No Extensions.*

***Exceptions:**

1. *Username or password is incorrect.*
2. *There is no Internet connection.*

Concurrent Uses: *None*

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

Decision Support

***Frequency:** *This use case will be performed at least once per application use.*

***Criticality:** *This is required for all transactions.*

***Risk:** *Low. This use case is simple to be implemented.*

Constraints:

- **Usability:** System operation not handled directly by the user.
- **Reliability:** Mean time to failure – 1% failure for every 1 week of use.
- **Performance:** Request should be handled in less than one (1) second, if no other requests exist.
- **Supportability:** The command must be properly handled by NCB.
- **Implementation:** Must be implemented in Java.

Modification History: - v1.10

***Owner:** Eduardo Monteiro

***Initiation date:** 09/09/2007

***Date last modified:** 09/22/2007

Use Case Logout

***Use Case ID:** *UCM_02 – Logout.*

Use Case Level: *Functional sub-use case.*

***Scenario:** *A control script is received from the Synthesis Engine containing the Logout("a") command.*

- **Actor:** *Synthesis Engine (SE), Network Communication Broker (NCB).*
- **Pre-conditions:**

1. *There exists a session with a logged in user.*

- **Description:**

  Trigger: *Use case begins when the Synthesis Engine requests a logout through the control script.*

  The system responds by:

1. *UCM calls the Logout function.*
2. *UCM receives an answer from the NCB stating if the Logout was successful or not.*

- **Relevant requirements:** *none.*

- **Post-conditions:** *The user is logged out.*

***Alternative Courses of Action** *No alternate action.*

Extensions: *No Extensions.*

***Exceptions:**

1. *There is no session.*
2. *There is no Internet connection.*

Concurrent Uses: *None*

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

Decision Support

***Frequency:** *This use case will be performed at least once per application use.*

***Criticality:** *This is not required. The system can be shutdown without a logout command.*

***Risk:** *Low. This use case is simple to be implemented.*

Constraints:

- **Usability:** System operation not handled directly by the user.
- **Reliability:** Mean time to failure – 1% failure for every 1 week of use.
- **Performance:** Request should be handled in less than one (1) second, if no other requests exist.
- **Supportability:** The command must be properly handled by NCB.
- **Implementation:** Must be implemented in Java.

Modification History: - v1.10

***Owner:** Eduardo Monteiro

***Initiation date:** 09/09/2007

***Date last modified:** 09/22/2007

Use Case Create Connection

***Use Case ID:** *UCM_03 – Create Connection.*

Use Case Level: *Functional sub-use case.*

***Scenario:** *A control script is received from the Synthesis Engine containing the createConnection("c1") command.*

- **Actor:** *Synthesis Engine (SE), Network Communication Broker (NCB).*
- **Pre-conditions:**

1. *There exists a session with a logged in user.*

- **Description:**

  Trigger: *Use case begins when the Synthesis Engine requests a Create Connection through the control script.*

  The system responds by:

1. *UCM calls the Create Connection function.*
2. *UCM receives an answer from the NCB, stating if the Create Connection command was successful or not, as well as a session ID.*

- **Relevant requirements:** *none.*
- **Post-conditions:** *A connection is created.*

***Alternative Courses of Action** *No alternate action.*

Extensions: *No Extensions.*

***Exceptions:**

1. *The user is not logged in.*
2. *There is no Internet connection.*

Concurrent Uses: *None*

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

Decision Support

***Frequency:** *This use case will be performed at least once per application use.*

***Criticality:** *This is an important use case. Without it, no communications can be created.*

***Risk:** *Low. This use case is simple to be implemented.*

Constraints:

- **Usability:** System operation not handled directly by the user.

- **Reliability:** Mean time to failure – 1% failure for every 1 week of use.
- **Performance:** Request should be handled in less than one (1) second, if no other request exist.
- **Supportability:** The command must be properly handled by NCB.
- **Implementation:** Must be implemented in java.

Modification History: - v1.10

***Owner:** Eduardo Monteiro

***Initiation date:** 09/09/2007

***Date last modified:** 09/22/2007

Use Case Decline Connection

***Use Case ID:** *UCM_04 – Decline Connection.*

Use Case Level: *Functional sub-use case.*

***Scenario:** *A control script is received from the Synthesis Engine containing the declineConnection("c1", "b", "a") command.*

- **Actor:** *Synthesis Engine (SE), Network Communication Broker (NCB).*
- **Pre-conditions:**

1. *There exists a session with a logged in user.*

- **Description:**

  Trigger: *Use case begins when the Synthesis Engine requests a Decline Connection through the control script.*

  The system responds by:

1. *UCM calls the Decline Connection function.*
2. *UCM receives an answer from the NCB, stating if the Decline Connection command was successful or not.*

- **Relevant requirements:** *none.*
- **Post-conditions:** *A connection is declined.*

***Alternative Courses of Action** *No alternate action.*

Extensions: *No Extensions.*

***Exceptions:**

1. *The user is not logged in.*

2. *There is no Internet connection.*

Concurrent Uses: *None*

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

Decision Support

***Frequency:** *This use case will be performed at least once per application use.*

***Criticality:** *This is an important use case. Without it, no communications can be declined.*

***Risk:** *Low. This use case is simple to be implemented.*

Constraints:

- **Usability:** System operation not handled directly by the user.
- **Reliability:** Mean time to failure – 1% failure for every 1 week of use.
- **Performance:** Request should be handled in less than one (1) second, if no other requests exist.
- **Supportability:** The command must be properly handled by NCB.
- **Implementation:** Must be implemented in Java.

Modification History: - v1.10

***Owner:** Eduardo Monteiro

***Initiation date:** 09/09/2007

***Date last modified:** 09/22/2007

Use Case Add Participant

***Use Case ID:** *UCM_05 – Add Participant*

Use Case Level: *Functional sub-use case.*

***Scenario:** *A control script is received from the Synthesis Engine containing a command to add a participant to the active connection. The control script contains information to add participant "b" to the current existing connection.*

- **Actor:** *Synthesis Engine (SE), Network Communication Broker (NCB).*

- **Pre-conditions:**

1. *The system has successfully performed the login event from SE and it is logged in.*
2. *The system has successful created a connection.*

- **Description:**

  Trigger: *Use case begins when the Synthesis Engine requests the addition of a participant via a control script.*

  The system responds by:

  1. *UCM obtains the session ID from the connection.*
  2. *UCM then adds the participants from into NCB's party list.*
  3. *Use case ends when every participant has been added.*

- **Relevant requirements:** *none.*
- **Post-conditions:** *The number in the NCB participant list has increased by one.*

***Alternative Courses of Action** *No alternate action.*

Extensions: *No Extensions.*

***Exceptions:**

1. *No session can be found.*
2. *The participant could not be added.*

Concurrent Uses: *None*

***Related Use Cases:** *UCM_01 – Log in, UCM_03 – Create Connection.*

Decision Support

***Frequency:** *This use case will be performed at least once per application use. This will occur every time that a request from the Synthesis Engine is made to add one or more participants to the connection. Expected times of executions 100 uses per session.*

***Criticality:** *High. This is required for anytime that the system requests the addition of participants.*

***Risk:** *Low. This use case is performed by a layer that is isolated from any system resource.*

Constraints:

- **Usability:** System operation not handled directly by the user.
- **Reliability:** Mean time to failure – 1% failure for every 100 executions.
- **Performance:** Request should be handled in less than one (1) second, if no other requests exist.
- **Supportability:** The command must be properly handled by NCB.

- **Implementation:** Must be implemented in Java.

Modification History: - v1.10

***Owner:** Frank Hernandez

***Initiation date:** 09/04/2007

***Date last modified:** 09/22/2007

Use Case Remove Participant

***Use Case ID:** *UCM_06 – Remove Participant*

Use Case Level: *Functional sub-use case.*

***Scenario:** *A control script is received from the Synthesis Engine containing a command to remove a participant to the active connection. A script containing the command to remove the participant "b" is passed by SE.*

- **Actor:** *Synthesis Engine (SE), Network Communication Broker (NCB).*
- **Pre-conditions:**

1. *The system has successfully performed the login event from SE and it is logged in.*
2. *The system has successfully created a connection.*
3. *The system has successfully added a participant.*

- **Description:**

  Trigger: *Use case begins when the Synthesis Engine request the removal of a participant via a control script.*

  The system responds by:

  1. *UCM obtains the session ID from the connection.*
  2. *Checks that the participant is in NCB's party list.*
  3. *UCM then removes the participant from the list into NCB's party list.*
  4. *Use case ends when every participant has been removed.*

- **Relevant requirements:** *none.*
- **Post-conditions:** *The number in the NCB participant list has decreased by one.*

***Alternative Courses of Action** *No alternate action.*

Extensions: *No Extensions.*

***Exceptions:**

1. *No session can be found.*
2. *The participant could not be added.*

Concurrent Uses: *None*

***Related Use Cases:** *UCM_01 – Log in, UCM_03 – Create Connection, UCM_05 – Add Participant.*

Decision Support

***Frequency:** *This use case will be performed at least once per application use. This will occur every time that a request from the Synthesis Engine is made to remove one or more participants to the connection. Expected times of executions 100 uses per session.*

***Criticality:** *High. This is required for anytime that the system requests the removal of participants.*

***Risk:** *Low. This use case is performed by a layer that is isolated from any system resource.*

Constraints:

- **Usability:** System operation not handled directly by the user.
- **Reliability:** Mean time to failure – 1% failure for every 1000 executions.
- **Performance:** Request should be handled in less than one (1) second, if no other requests exist.
- **Supportability:** The command must be properly handled by NCB.
- **Implementation:** Must be implemented in Java.

Modification History: - v1.10

***Owner:** Frank Hernandez

***Initiation date:** 09/04/2007

***Date last modified:** 09/22/2007

Use Case Send Media

***Use Case ID:** *UCM_07 – Send Media*

Use Case Level: *Functional sub-use case.*

***Scenario:** *A control script is received from the Synthesis Engine containing a command to send a medium to participant B in the connection. The command is passed with parameters of media "audio".*

- **Actor:** *Synthesis Engine (SE), Network Communication Broker (NCB).*
- **Pre-conditions:**

1. *The system has successfully performed the login event from SE and it is logged in.*
2. *The system has successfully created a connection.*
3. *The system has successfully added a participant.*
4. *The system has successfully enabled the media initiator.*

- **Description:**

  Trigger: *Use case begins when the Synthesis Engine requests the sending of a medium via a control script.*

  The system responds by:

  1. *UCM finds the participants using the connection ID.*
  2. *UCM sends the specified medium to every participant in NCB's party list.*
  3. *Use case ends when media has been sent.*

- **Relevant requirements:** *none.*
- **Post-conditions:** *The new medium has been sent to all participants.*

***Alternative Courses of Action** *No alternate action.*

Extensions: *No Extensions.*

***Exceptions:**

1. *No session can be found.*
2. *No participant could be found.*
3. *No medium could be found.*

Concurrent Uses: *None*

***Related Use Cases:** *UCM_01 – Log in, UCM_03 – Create Connection, UCM_05 – Add Participant, UCM_24 – Enable Media Initiator.*

Decision Support

***Frequency:** *This use case will be performed at least once per application use. This will occur every time that a request from the Synthesis Engine is made to send a medium. Expected times of executions 100 uses per session.*

***Criticality:** *High. This is required for anytime that the system requests the sending of a medium.*

***Risk:** *Low. This use case is performed by a layer that is isolated from any system resource.*

Constraints:

- **Usability:** System operation not handled directly by the user.
- **Reliability:** Mean time to failure – 1% failure for every 1000 executions use.
- **Performance:** Request should be handled in less than one (1) second, if no other requests exist.
- **Supportability:** The command must be properly handled by NCB.
- **Implementation:** Must be implemented in Java.

Modification History: - v1.10

***Owner:** Frank Hernandez

***Initiation date:** 09/04/2007

***Date last modified:** 09/22/2007

Use Case Send Form

***Use Case ID:** *UCM_08 – Send Form*

Use Case Level: *Functional sub-use case.*

***Scenario:** *A control script is received from the Synthesis Engine containing a command to send a form to participant "b" in the connection "c1".*

- **Actor:** *Synthesis Engine (SE), Network Communication Broker (NCB).*
- **Pre-conditions:**

1. *The system has successfully performed the login event from SE and it is logged in.*
2. *The system has successfully created a connection.*
3. *The system has successfully added a participant.*

- **Description:**

  Trigger: *Use case begins when the Synthesis Engine requests the sending of a form via a control script.*

  The system responds by:

  1. *UCM finds the participants using the connection ID.*

2. *UCM sends the specified form to every participant in NCB's party list.*
3. *Use case ends when form has been sent.*

- **Relevant requirements:** *none.*
- **Post-conditions:** *The new form has been sent to all participants.*

***Alternative Courses of Action** *No alternate action.*

Extensions: *No Extensions.*

***Exceptions:**

1. *No session can be found.*
2. *No participant could be found.*
3. *No form could be found.*

Concurrent Uses: *None*

***Related Use Cases:** *UCM_01 – Log in, UCM_03 – Create Connection, UCM_05 – Add Participant.*

Decision Support

***Frequency:** *This use case will be performed at least once per application use. This will occur every time that a request from the Synthesis Engine is made to send a form. Expected to be executed 100 per session.*

***Criticality:** *This is required for anytime that the system requests the sending of a form.*

***Risk:** *Low. This use case is performed by a layer that is isolated from any system resource.*

Constraints:

- **Usability:** System operation not handled directly by the user.
- **Reliability:** Mean time to failure – 1% failure for every 1000 executions.
- **Performance:** Request should be handled in less than one (1) second, if no other requests exist.
- **Supportability:** The command must be properly handled by NCB.
- **Implementation:** Must be implemented in Java.

Modification History: - v1.10

***Owner:** Frank Hernandez

***Initiation date:** 09/04/2007

***Date last modified:** 09/22/2007

Use Case Create 2-way Audio

***Use Case ID:** *UCM_9 – Create 2-way Audio*

Use Case Level: *Functional sub-use case.*

***Scenario:** *A control script is received from the Synthesis Engine containing a command to create 2-way audio with participant John Doe for the sessionID "ses!23" , that is, a connection between sender side and receiver side.*

- **Actor:** *Synthesis Engine (SE), Network Communication Broker (NCB).*

  Pre-conditions:

  1. *The script received from the interpreter is in the legal format.*
  2. *The macro for the script is found in the repository.*
  3. *There are no network failures in the communication.*

- **Description:**

  Trigger: *Use case begins when the Synthesis Engine requests the creation of 2-way audio between two users via a control script.*

  The system responds by:

  1. *UCM handles the login command in the control script for the sender. (See Use Case UCM_01 – Log in)*
  2. *UCM handles the login command in the control script for the receiver. (See Use Case UCM_01 – Log in)*
  3. *UCM handles the create connections command in the control script for the sender. (See Use Case UCM_03 – Create Connection)*
  4. *UCM handles the add participants command in the control script for the sender. (See Use Case UCM_05 – Add Participant)*
  5. *UCM handles the send schema command in the control script to for the sender. (See Use Case UCM_14 – Send Schema)*
  6. *UCM handles the send schema command in the control script to for the receiver. (See Use Case UCM_14 – Send Schema)*
  7. *UCM handles the media initiator command in the control script for the sender. (See Use Case UCM_16 – Enable Media Receiver)*
  8. *UCM handles the media receiver command in the control script for the receiver. (See Use Case UCM_24 – Enable Media Initiator)*

- **Relevant requirements:** *{In this section reference is made to any other requirements documents such as industry standards or government regulations.}*
- **Post-conditions:** *The Event handler creates the particular event and is logged out.*

***Alternative Courses of Action** *No alternate action.*

Extensions: *No Extensions.*

***Exceptions:**

1. *Unable to create an event by the Event handler.*
2. *Exception generated by the NCB after a network failure.*

Concurrent Uses: *None*

***Related Use Cases:** *UCM_01 – Log in , UCM_03 – Create Connection, UCM_05 – Add Participant, UCM_14 – Send Schema, UCM_16 – Enable Media Receiver, UCM_24 – Enable Media Initiator*

Decision Support

***Frequency:** *This use case will be performed every time when a user tries to create 2-way audio.*

***Criticality:** *This is required anytime that the system requests the creation of 2-way audio. This is an important use case as without it there would be no 2-way audio communication between two users.*

***Risk:** *High. This use case relies on several other use cases.*

Constraints:

- **Usability:** System operation not handled directly by the user.
- **Reliability:** Mean time to failure – 8% failure for every 24hrs use.
- **Performance:** Request should be handled in less than five (5) seconds, if no other requests exist.
- **Supportability:** The commands must be properly handled by Event handler.
- **Implementation:** Must be implemented in Java.

Modification History: - v1.10

***Owner:** Raidel Batista

***Initiation date:** 09/10/2007

***Date last modified:** 09/22/2007

Use Case Create Conference Audio

***Use Case ID:** *UCM_10 – Create Conference Audio*

Use Case Level: *Functional sub-use case.*

***Scenario:** *A control script is received from the Synthesis Engine containing a command to create conference audio with participants John Doe, Jane Doe and Mary Doe for the sessionID "ses!23" , that is, a connection between sender side and receiver side.*

- **Actor:** *Synthesis Engine (SE), Network Communication Broker (NCB).*

  Pre-conditions:

  1. *The script received from the interpreter is in the legal format.*
  2. *The macro for the script is found in the repository.*
  3. *There are no network failures in the communication.*

- **Description:**

  Trigger: *Use case begins when the Synthesis Engine requests the creation of conference audio between three users via a control script.*

  The system responds by:

  1. *UCM handles the login command in the control script for the sender. (See Use Case UCM_01 – Log in)*
  2. *UCM handles the login command in the control script for receiver one. (See Use Case UCM_01 – Log in)*
  3. *UCM handles the login command in the control script for receiver two. (See Use Case UCM_01 – Log in)*
  4. *UCM handles the create connections command in the control script for the sender. (See Use Case UCM_03 – Create Connection)*
  5. *UCM handles the add participants command in the control script for the sender. (See Use Case UCM_05 – Add Participant)*
  6. *UCM handles the send schema command in the control script to for the sender. (See Use Case UCM_14 – Send Schema)*
  7. *UCM handles the send schema command in the control script to for receiver one. (See Use Case UCM_14 – Send Schema)*
  8. *UCM handles the send schema command in the control script to for receiver two. (See Use Case UCM_14 – Send Schema)*
  9. *UCM handles the media initiator command in the control script for the sender. (See Use Case UCM_16 – Enable Media Receiver)*

10. *UCM handles the media receiver command in the control script for receiver one. (See Use Case UCM_24 – Enable Media Initiator)*
11. *UCM handles the media receiver command in the control script for receiver two. (See Use Case UCM_24 – Enable Media Initiator)*

- **Relevant requirements:** *{In this section reference is made to any other requirements documents such as industry standards or government regulations.}*
- **Post-conditions:** *The Event handler creates the particular event and is logged out.*

***Alternative Courses of Action** *No alternate action.*

Extensions: *No Extensions.*

***Exceptions:**

1. *Unable to create an event by the Event handler.*
2. *Exception generated by the NCB after a network failure.*

Concurrent Uses: *None*

***Related Use Cases:** *UCM_01 – Log in , UCM_03 – Create Connection, UCM_05 – Add Participant, UCM_14 – Send Schema, UCM_16 – Enable Media Receiver, UCM_24 – Enable Media Initiator*

Decision Support

***Frequency:** *This use case will be performed every time when a user tries to create conference audio at least 1 time per session.*

***Criticality:** *This is required anytime that the system requests the creation of conference audio. This is an important use case as without this there will be no conference audio communication between three users.*

***Risk:** *High. This use case relies on several other use cases.*

Constraints:

- **Usability:** System operation not handled directly by the user.
- **Reliability:** Mean time to failure – 8% failure for every 24hrs use.
- **Performance:** Request should be handled in less than five (5) seconds, if no other requests exist.
- **Supportability:** The commands must be properly handled by Event handler.
- **Implementation:** Must be implemented in Java.

Modification History: - v1.10

***Owner:** Raidel Batista

***Initiation date:** 09/10/2007

***Date last modified:** 09/22/2007

Use Case Create 2-way Video

***Use Case ID:** *UCM_11 – Create 2-way Video*

Use Case Level: *Functional sub-use case.*

***Scenario:** *A control script is received from the Synthesis Engine containing a command to create 2-way video with participant John Doe for the sessionID "ses!23" , that is, a connection between sender side and receiver side.*

- **Actor:** *Synthesis Engine (SE), Network Communication Broker (NCB).*

  Pre-conditions:

  1. *The script received from the interpreter is in the legal format.*
  2. *The macro for the script is found in the repository.*
  3. *There are no network failures in the communication.*

- **Description:**

  Trigger: *Use case begins when the Synthesis Engine requests the creation of 2-way video between two users via a control script.*

  The system responds by:

  1. *UCM handles the login command in the control script for the sender. (See Use Case UCM_01 – Log in)*
  2. *UCM handles the login command in the control script for the receiver. (See Use Case UCM_01 – Log in)*
  3. *UCM handles the create connections command in the control script for the sender. (See Use Case UCM_03 – Create Connection)*
  4. *UCM handles the add participants command in the control script for the sender. (See Use Case UCM_05 – Add Participant)*
  5. *UCM handles the send schema command in the control script to for the sender. (See Use Case UCM_14 – Send Schema)*
  6. *UCM handles the send schema command in the control script to for the receiver. (See Use Case UCM_14 – Send Schema)*
  7. *UCM handles the media initiator command in the control script for the sender. (See Use Case UCM_16 – Enable Media Receiver)*

8. *UCM handles the media receiver command in the control script for the receiver. (See Use Case UCM_24 – Enable Media Initiator)*

- **Relevant requirements:** *{In this section reference is made to any other requirements documents such as industry standards or government regulations.}*
- **Post-conditions:** *The Event handler creates the particular event and is logged out.*

***Alternative Courses of Action** *No alternate action.*

Extensions: *No Extensions.*

***Exceptions:**

1. *Unable to create an event by the Event handler.*
2. *Exception generated by the NCB after a network failure.*

Concurrent Uses: *None*

***Related Use Cases:** *UCM_01 – Log in , UCM_03 – Create Connection, UCM_05 – Add Participant, UCM_14 – Send Schema, UCM_16 – Enable Media Receiver, UCM_24 – Enable Media Initiator*

Decision Support

***Frequency:** *This use case will be performed every time when a user tries to create 2-way video, at least 1 per session.*

***Criticality:** *This is required anytime that the system requests the creation of 2-way video. This is an important use case as without this there will be no 2-way video communication between two users.*

***Risk:** *High. This use case relies on several other use cases.*

Constraints:

- **Usability:** System operation not handled directly by the user.
- **Reliability:** Mean time to failure – 8% failure for every 24hrs use.
- **Performance:** Request should be handled in less than five (5) seconds, if no other requests exist.
- **Supportability:** The commands must be properly handled by Event handler.
- **Implementation:** Must be implemented in Java.

Modification History: - v1.10

***Owner:** Raidel Batista

***Initiation date:** 09/10/2007

***Date last modified:** 09/22/2007

Use Case Create Conference Video

***Use Case ID:** *UCM_12 – Create Conference Video*

Use Case Level: *Functional sub-use case.*

***Scenario:** *A control script is received from the Synthesis Engine containing a command to create conference video with participants John Doe, Jane Doe and Mary Doe for the sessionID "ses!23" , that is, a connection between sender side and receiver side.*

- **Actor:** *Synthesis Engine (SE), Network Communication Broker (NCB).*

  Pre-conditions:

  1. *The script received from the interpreter is in the legal format.*
  2. *The macro for the script is found in the repository.*
  3. *There are no network failures in the communication.*

- **Description:**

  Trigger: *Use case begins when the Synthesis Engine requests the creation of conference video between three users via a control script.*

  The system responds by:

  1. *UCM handles the login command in the control script for the sender. (See Use Case UCM_01 – Log in)*
  2. *UCM handles the login command in the control script for receiver one. (See Use Case UCM_01 – Log in)*
  3. *UCM handles the login command in the control script for receiver two. (See Use Case UCM_01 – Log in)*
  4. *UCM handles the create connections command in the control script for the sender. (See Use Case UCM_03 – Create Connection)*
  5. *UCM handles the add participants command in the control script for the sender. (See Use Case UCM_05 – Add Participant)*
  6. *UCM handles the send schema command in the control script to for the sender. (See Use Case UCM_14 – Send Schema)*
  7. *UCM handles the send schema command in the control script to for receiver one. (See Use Case UCM_14 – Send Schema)*

8. *UCM handles the send schema command in the control script to for receiver two. (See Use Case UCM_14 – Send Schema)*
9. *UCM handles the media initiator command in the control script for the sender. (See Use Case UCM_16 – Enable Media Receiver)*
10. *UCM handles the media receiver command in the control script for receiver one. (See Use Case UCM_24 – Enable Media Initiator)*
11. *UCM handles the media receiver command in the control script for receiver two. (See Use Case UCM_24 – Enable Media Initiator)*

- **Relevant requirements:** *{In this section reference is made to any other requirements documents such as industry standards or government regulations.}*
- **Post-conditions:** *The Event handler creates the particular event and is logged out.*

***Alternative Courses of Action** *No alternate action.*

Extensions: *No Extensions.*

***Exceptions:**

1. *Unable to create an event by the Event handler.*
2. *Exception generated by the NCB after a network failure.*

Concurrent Uses: *None*

***Related Use Cases:** *UCM_01 – Log in , UCM_03 – Create Connection, UCM_05 – Add Participant, UCM_14 – Send Schema, UCM_16 – Enable Media Receiver, UCM_24 – Enable Media Initiator*

Decision Support

***Frequency:** *This use case will be performed every time when a user tries to create conference video, at least 1 per session.*

***Criticality:** *This is required anytime that the system requests the creation of conference video. This is an important use case as without this there will be no conference video communication between three users.*

***Risk:** *High. This use case relies on several other use cases.*

Constraints:

- **Usability:** System operation not handled directly by the user.
- **Reliability:** Mean time to failure – 8% failure for every 24hrs use.
- **Performance:** Request should be handled in less than five (5) seconds, if no other requests exist.

- **Supportability:** The commands must be properly handled by Event handler.
- **Implementation:** Must be implemented in Java.

Modification History: - v1.10

***Owner:** Raidel Batista

***Initiation date:** 09/10/2007

***Date last modified:** 09/22/2007

Use Case Send Demand Form

***Use Case ID:** *UCM_13 – Send Demand Form*

Use Case Level: *Functional sub-use case.*

***Scenario:** *A control script is received from the Synthesis Engine containing a command to send a demanded, or requested, form to a certain receiver. The script contains sendDemandForm("c1","form1", www.cs.fiu.edu);*

- **Actor:** *Synthesis Engine (SE), Network Communication Broker (NCB).*
- **Pre-conditions:**

1. *The system has successfully created a connection.*
2. *A form with the requested ID exists.*
3. *The form is being requested at a receiver side.*

- **Description:**

  Trigger: *Use case begins when the Synthesis Engine requests a demanded form to be sent via a control script.*

  The system responds by:

  1. *UCM obtains the connection ID*
  2. *UCM obtains the information of the user requesting the form*
  3. *UCM obtains the form ID*
  4. *UCM obtains the mediumURL*
  5. *UCM then sends the demanded form*
  6. *Use case ends when the form is sent*

- **Post-conditions:** *The other party successfully received the form.*

***Alternative Courses of Action:**

1. *In step D3., if the formID is invalid, the operation is cancelled.*
2. *In step D4, if the medium is not available, the operation is cancelled.*

Extensions: *No Extensions.*

***Exceptions:**

1. *No session can be found.*
2. *The demand form cannot be sent.*

Concurrent Uses: *None*

***Related Use Cases:** *UCM_08 – SendForm.*

Decision Support

***Frequency:** *This will occur every time that a request from the Synthesis Engine is made to send a schema based on a request from another party at least one time per session.*

***Criticality:** *High. This is required for anytime that the system requests to send a demanded schema.*

***Risk:** *Low. This use case is performed by a layer that is isolated from any system resource.*

Constraints:

- **Usability:** System operation not handled directly by the user.
- **Reliability:** Mean time to failure – 5% failure for every 24 hrs use.
- **Performance:** Request should be handled in less than five seconds, if no other requests exist and considering the number of participants in the connection.
- **Supportability:** The command must be properly handled by NCB.
- **Implementation:** Must be implemented in Java.

Modification History: - v1.2

***Owner:** Marylurdys Hernandez

***Initiation date:** 09/17/2007

***Date last modified:** 09/22/2007

Use Case Send Schema

***Use Case ID:** *UCM_14 – Send Schema*

Use Case Level: *Functional sub-use case.*

***Scenario:** *A control script is received from the Synthesis Engine containing a command to sendSchema("c1","a","b","control_xcml", "data_xcml").*

- **Actor:** *Synthesis Engine (SE), Network Communication Broker (NCB).*
- **Pre-conditions:**

1. *The system has successfully performed the login event from SE and it is logged in*
2. *The system has successfully created a connection and a session IDs*
3. *The system has added participants to the connection such as there are at least two participants in the connection*

- **Description:**

  Trigger: *Use case begins when the Synthesis Engine requests to send a schema via a control script.*

  The system responds by:

  1. *UCM obtains the session ID from the connection*
  2. *NCB sends the control schema to all participants in the specified connection*
  3. *NCB sends the data schema to all participants in the specified connection*

- **Post-conditions:** *The initiator receives in return a schema from all participants.*

***Alternative Courses of Action** *No alternate action*

Extensions: *No Extensions.*

***Exceptions:**

1. *The session ID of the connection is null*
2. *The control schema is null*
3. *The data schema is null*

Concurrent Uses: *None*

***Related Use Cases:** *UCM_01–Login, UCM_03–CreateConnection, UCM_05–Add Participant*

Decision Support

***Frequency:** *This will occur every time that a request from the Synthesis Engine is made to send a schema on average of two per session.*

***Criticality:** *This is required for anytime that the system requests to send a schema.*

***Risk:** *Low. This use case is performed by a layer that is isolated from any system resource.*

Constraints:

- **Usability:** System operation not handled directly by the user.
- **Reliability:** Mean time to failure – 10% failure for every 24 hrs use.
- **Performance:** Request should be handled in less than five seconds, if no other requests exist and considering the number of participants in the connection.
- **Supportability:** The command must be properly handled by NCB.
- **Implementation:** Must be implemented in Java.

Modification History: - v1.2

***Owner:** Marylurdys Hernandez

***Initiation date:** 09/17/2007

***Date last modified:** 09/22/2007

Use Case Receive Schema

***Use Case ID:** *UCM_15 – Receive Schema*

Use Case Level: *Functional sub-use case.*

***Scenario:** *A control script is received from the Synthesis Engine containing a command to receive a schema.*

- **Actor:** *Synthesis Engine (SE), Network Communication Broker (NCB).*
- **Pre-conditions:**

1. *The system has successfully performed the login event from SE and it is logged in*
2. *The system has completed a negotiation to receive a schema*

- **Description:**

  Trigger: *Use case begins when the Synthesis Engine sends a schema via a control script to this user.*

  The system responds by:

  1. *NCB generates a NotifySchemaReceived event that will be received by event handler and passed to UCM manager.*
  2. *UCM manager generates new event and signals the SE.*

- **Post-conditions:** *The invitation is accepted or rejected.*

***Alternative Courses of Action** *No alternate action.*

Extensions: *No Extensions.*

***Exceptions:**

Concurrent Uses: *None*

***Related Use Cases:** *UCM_14 – SendSchema.*

Decision Support

***Frequency:** *This will occur every time that the system has previously finished negotiations for sending a schema and the receiver is ready to accept it, on average of 2 per session.*

***Criticality:** *This is required for anytime that the system is receiving a schema.*

***Risk:** *Low. This use case is performed by a layer that is isolated from any system resource.*

Constraints:

- **Usability:** System operation not handled directly by the user.
- **Reliability:** Mean time to failure – 4% failure for every 24 hrs use.
- **Performance:** Request should be handled in less than two seconds.
- **Supportability:** The command must be properly handled by NCB.
- **Implementation:** Must be implemented in Java.

Modification History: - v1.2

***Owner:** Marylurdys Hernandez

***Initiation date:** 09/17/2007

***Date last modified:** 09/22/2007

Use Case Enable Media Receiver

***Use Case ID:** *UCM_16 – Enable Media Receiver*

Use Case Level: *Functional sub-use case.*

***Scenario:** *A control script is received from the Synthesis Engine containing the command enableMediaReceiver("c1", "audio") for the connection c1.*

- **Actor:** *Synthesis Engine (SE), Network Communication Broker (NCB).*
- **Pre-conditions:**

1. *The system has successfully performed the login event from SE and it is logged in*
2. *The system has successful created a mapped connection and a session IDs*
3. *The system has added participants to the connection, such as there are at least two participants in the connection*
4. *An initiator has sent a schema to all participants in the connection*
5. *This user has accepted the connection*
6. *This user sends back the final schema*
7. *The initiator sends the final schema back*

- **Description:**

  Trigger: *Use case begins when the last schema two received schemas are consistent. The Synthesis Engine requests the enablement of media receiver via a control script.*

  The system responds by:

  1. *UCM obtains the session ID from the connection.*
  2. *UCM enables the specified list of media to be received.*

- **Relevant requirements:** *{In this section reference is made to any other requirements documents such as industry standards or government regulations.}*
- **Post-conditions:** *Media receiving is enabled in the receiver side.*

***Alternative Courses of Action** *No alternate action.*

Extensions: *No Extensions.*

***Exceptions:**

1. *The session ID is null.*

Concurrent Uses: *None*

***Related Use Cases:** *UCM_14 – SendSchema, UCM_24 – EnableMediaInitiator.*

Decision Support

***Frequency:** *This will occur every time that a request from the Synthesis Engine is made to enable media receiver at least 1 per session.*

***Criticality:** *This is required for anytime that the system requests to enable media receiver.*

***Risk:** *Low. This use case is performed by a layer that is isolated from any system resource.*

Constraints:

- **Usability:** System operation not handled directly by the user.
- **Reliability:** Mean time to failure – 15% failure for every 24 hrs use.
- **Performance:** Request should be handled in less than three seconds, if no other requests exist and considering the number of participants in the connection.
- **Supportability:** The command must be properly handled by NCB.
- **Implementation:** Must be implemented in Java.

Modification History: - v1.2

***Owner:** Marylurdys Hernandez

***Initiation date:** 09/17/2007

***Date last modified:** 09/22/2007

Use Case Load Macro

***Use Case ID:** *UCM_17-Load Macro*

Use Case Level: *Functional sub-use case*

***Scenario:** *A control script including "removeParticipant" is received from the Synthesis Engine through the UCM-SE interface. The UCM manager then recognizes this control script and loads the defined macros from the local repository into memory. The "removeParticipant" is now ready for deployment and execution.*

- **Actor: :** *Synthesis Engine (SE), Network Communication Broker (NCB).*
- **Pre-conditions:**
  1. *The macro definition for this control script is predefined and stored in the local repository*
  2. *The control script is both syntactically and semantically valid .*
- **Description:**

  Trigger: *Use case begins when a control script from the synthesis engine is passed down to UCM for execution*

The system responds by.

1. *UCM checks the type of the control script to be executed*
2. *UCM load the corresponding macro of the control script into memory*
3. *Use case ends when the macro to be executed is in memory ready for execution.*

- **Relevant requirements:** *To increase portability, this use case should in no way depend on actual macros, and the location of the macros in the repository.*
- **Post-conditions:** *Macro is loaded into the memory and ready for execution*

***Alternative Courses of Action** *No alternate action.*

Extensions: *None*

***Exceptions:** *No macro found in the local repository for this control script*

Concurrent Uses: *None*

***Related Use Cases:** *the SaveMacroInstance use case*

Decision Support

***Frequency:** *This use case will be performed at least ten times per application use, for the simplest case of communication. It will actually occur every time a control script from the Synthesis Engine is sent down for execution.*

***Criticality:** *The absence of this use case will not only prevent the user from logging in to the system, but also prevent any communication to be established. It is required for the correct functioning of UCM.*

***Risk: :** *Low. This use case is performed by a layer that is isolated from any system resource.*

Constraints:

- **Performance:** The loading process should be done within 10 milliseconds. However, loading macros must be done sequentially because only one script is executed at a time.
- **Supportability:** This use case should not depend on the actual scripts and macros. It should be extensible to repositories populated with large numbers of macros.
- **Interface:** This use case should provide only one interface: loadMacro to the synthesis engine developer, and hide the internal operation.
- **Implementation:** Must be implemented in java.
- **Operation:** The running UCM is invoked by SE and supported by NCB.

Modification History – *v1.2*

***Owner:** Guangqiang Zhao

***Initiation date:** 09/08/2007

***Date last modified:** 09/21/2007

Use Case Save Macro Instance

Use Case ID: *UCM_18-Save Macro Instance*

Use Case Level: Functional Sub-Use Case

***Scenario:** *UCM has requested to save the instance for Macro login with user name "Burke" and pw = "1234". UCM Manager then saves that macro instance into the local repository. UCM manager also maps the login script with the macro instance provided.*

- **Actor:** *Macro developer, Local repository*
- **Pre-conditions:** *The macro definition is already defined and ready for storage.*
- **Description:**

  Trigger: *The user initiates an action requesting to save the macro instance*

  The system responds by:

  1. *Saving the macro instance into the local repository as a string or a text file*
  2. *Register this macro in the local repository by adding a script-to-macro mapping*

- **Relevant requirements:** *There is not macro saved in the local repository for this particular control script*
- **Post-conditions:** *The macro is saved into the local repository and the script-to-macro mapping is saved as well.*

***Alternative Courses of Action** *: No alternative actions*

Extensions:

1. *These is already a macro instance defined for this particular control script, in this case the system should either prompt to change to another control script or would not allow this action.*

***Exceptions:**

1. *The macro instance is not successfully saved into the local repository, this exceptions has to be handled by the Exception Handler.*

Concurrent Uses: *None*

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

Decision Support

***Frequency:** *This use case will be performed intensively before the first application use. This will occur every time a new macro instance for a control script is created and need to be saved. And then, as UCM supports more control scripts, it will be executed per control script.*

***Criticality:** *This is required for anytime a macro definition for a new control script need to be saved into the local repository*

***Risk:** *Low. This use case is performed by a layer that is isolated from any system resource.*

Constraints:

- **Supportability:** This use case should be able to save any form of macro instances.
- **Interface:** This macro should only have one interface: saveMacro() to the macro developer.
- **Implementation:** This use case should be able to save the macro in all the mainstream file systems.
- **Performance:** This use case should be handled within 10 milliseconds.
- **Implementation:** Must be implemented in java.

Modification History – *v1.2*

***Owner:** Guangqiang Zhao

***Initiation date:** 09/08/2007

***Date last modified:** 09/21/2007

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

Use Case Create Exception

***Use Case ID:** *UCM_19-Create Exception*

Use Case Level: *Functional sub-use case*

***Scenario:** *NCB notifies a ”mediumNotSent” exception to the UCM interpreter during the execution of the “sendMedium” command. The interpreter then notifies the Exception handler about this exception. The “mediumNotSent” exception is then handled by the Exception handler.*

- **Actor:** *Network Communication Broker (NCB).*
- **Pre-conditions*:*** *This exception type must be able to be handled by the Exception Handler.*
- **Description:**

  Trigger: *UCM identifies an execution exception and is ready to create an exception*

  The system responds by

  1. *Check the exception type and exception information returned by the NCB*
  2. *Wrap the exception information as a parameter of the notification.*
  3. *Report the new Exception to the exception handler.*
- **Relevant requirements:** *None*
- **Post-conditions:** *The Exception is notified to the exception handler.*

***Alternative Courses of Action** *: No alternative actions*

Extensions: *None*

***Exceptions:** *None*

Concurrent Uses: *None*

***Related Use Cases:** the *HandelException use case*

Decision Support

***Frequency:** *This use case will be performed at least once per application use. It is executed whenever an exception is identified and need to be created for further handling.*

***Criticality:** *It is critically important for the correct functioning of the system.*

***Risk:** *Low. This use case is performed by a layer that is isolated from any system resource.*

Constraints:

- **Usability:** System operation not handled directly by the user.

- **Performance:** Exception should be caught by the system within 10 milliseconds.
- **Reliability:** Mean time to failure – 5% failure for every 24 hrs use.
- **Supportability:** This use case should be able to identify the current 15 exceptions that might result from the runtime execution.
- **Implementation:** Must be implemented in Java.

Modification History – *v1.2*

***Owner:** Guangqiang Zhao

***Initiation date:** 09/09/2007

***Date last modified:** 09/21/2007

Use Case Handle Exception

***Use Case ID:** *UCM_20-Handle Exception*

Use Case Level: *Functional sub-use case*

***Scenario:** *The "loginException" occurs during the execution of some control script containing "login". The UCM interpreter notifies the exception handler about this exception. The exception handler notifies the Synthesis Engine about the login failure.*

Actor: *Synthesis Engine(SE)*

- **Pre-conditions:**
  1. *The exception is already created and caught by the exception handler.*
  2. *Policies for handling each exception are already defined.*
- **Description:**

  Trigger: *An exception is identified and reported to the Exception Handler.*

  The system responds by

  1. *Check the exception type and associated parameters of this exception*
  2. *Decide on what actions should take to fix this exception, by predefined policies.*
  3. *Handle this exception by either notifying the Synthesis engine or to fix up this exception inside UCM, either by retrying or using alternative policies.*

- **Relevant requirements:**
- **Post-conditions:** *The exception is caught and handled successfully*.

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

Extensions: *None*

***Exceptions:** *None*

Concurrent Uses: *None*

***Related Use Cases:** *The CreateException use case*

Decision Support

***Frequency:** *This use case will be performed at least once per application use . It is executed whenever an exception is caught.*

***Criticality:** *It is critically important to avoid system crash, therefore should be handled carefully..*

***Risk:** *Low. This use case is performed by a layer that is isolated from any system resource.*

Constraints:

- **Usability:** Should provide nice and informative exception messages to the user.
- **Reliability:** Mean time to failure – 5% failure for every 24 hrs use.
- **Performance:** Exceptions should be handled within 5 seconds, before user gets bored..
- **Supportability:** This use case should handle the current 15 exceptions that might result from the runtime execution.
- **Implementation:** Must be implemented in Java.
- **Interface:** This use case should only provide one method:handle() as an interface

---

Modification History – *v1.2*

***Owner:** Guangqiang Zhao

***Initiation date:** 09/08/2007

***Date last modified:** 09/12/2007

Use Case Create Event

***Use Case ID:** *UCM_21 – Create Event*

Use Case Level: *Functional sub-use case.*

***Scenario:** *Synthesis Engine sends a control script to UCM containing a command to create an event 'E' for the sessionID 's1' with the active connection 'c1'.*

- **Actor:** *Synthesis Engine (SE), Network Communication Broker (NCB).*
- **Pre-conditions:**
  1. *The script received form the interpreter is in the legal format.*
  2. *Participants are logged in and there is an active connection present.*
  3. *A session with sessionID is generated and currently running in process.*
- **Description:**

  Trigger: *Use case begins when the Synthesis Engine request the enabling of media via a control script.*

  The system responds by:

1. *UCM receives a message from the NCB to be passed on to the SE.*
2. *An event is created in response to the notification from the NCB and the events are queued.*
3. *UCM obtains the sessionID for the current connection session.*
4. *UCM calls the Create Event function to the SE, which creates a log for the particular event.*

- **Relevant requirements:** *{In this section reference is made to any other requirements documents such as industry standards or government regulations.}*
- **Post-conditions:** *The Event handler creates the particular event and is logged in the Local Repository.*

***Alternative Courses of Action** *No alternate action.*

Extensions: *No Extensions.*

***Exceptions:**

1. *Unable to create an event by the Event handler.*
2. *The macro for a script is not found in the repository.*
3. *There is a network failure in the communication.*

Concurrent Uses: *None*

***Related Use Cases:** *UCM_01 -- Login, UCM_05 – AddParticipant, UCM_03 – Create Connection.*

Decision Support

***Frequency:** *This use case will be performed every time a call is made from NCB to log a particular event. It will be initiated at least 10 times per session.*

***Criticality:** *High. This is an important use case as it logs the event and takes proper actions in case of exceptions or can rollback to the previous state in case of a system or network failure.*

***Risk:** *High. The system creates events for the purpose of saving them and loading them back whenever required.*

Constraints:

- **Usability:** System users are other subsystems(NCB and SE), which communicate through well-defined scripts and are easy to operate for other systems.
- **Reliability:** 10% failure rate is allowed for every 24hrs use.
- **Performance:** Requests should be handled in less than 2 minutes, if no other requests exist. Response time can be more depending on the number of events required to be created.
- **Supportability:** The command must be properly supported by Event handler of UCM.
- **Implementation:** Must be implemented in Java.

Modification History: - v1.2

***Owner:** Abhishek Bhattacharya

***Initiation date:** 09/10/2007

***Date last modified:** 09/23/2007

Use Case Load Event State

***Use Case ID:** *UCM_22 – Load Event*

Use Case Level: *Functional sub-use case.*

***Scenario:** *Synthesis Engine sends a control script to UCM containing a command to load an event 'E' for the sessionID 's1' with the active connection 'c1'.*

Actor: *Synthesis Engine (SE), Network Communication Broker (NCB).*

- **Pre-conditions:**

1. *Participants are logged in and there is an active connection present.*
2. *A session with sessionID is generated and currently running in process.*
3. *The script received form the interpreter is in the legal format.*
4. *There is a failure in the system, which requires to rollback by loading the saved event.*

- **Description:**

  Trigger: *Use case begins when the Synthesis Engine request the logged event to be loaded via a control script.*

  The system responds by:

  1. *Event Queue is loaded in the UCM in response to a system failure and the system needs to rollback to its previous state.*
  2. *UCM obtains the session ID from the connection session.*
  3. *UCM calls the Load Event function whenever the logged events are to be loaded.*

- **Relevant requirements:** *{In this section reference is made to any other requirements documents such as industry standards or government regulations.}*
- **Post-conditions:** *The Event handler loads the corresponding event state from the Local Repository.*

***Alternative Courses of Action** *No alternate action.*

Extensions: *No Extensions.*

***Exceptions:**

1. *Unable to load the event state from the Local Repository.*
2. *Not able to communicate with the NCB or SE.*

Concurrent Uses: *None*

***Related Use Cases:** *UCM_21 – Create Event, UCM_23 – Save Event State.*

Decision Support

***Frequency:** *This use case will be performed every time a call is made to load a particular logged event by the UCM Manager. It will be initiated at least 5 times per session whenever there is a system failure.*

***Criticality:** *High. This is important for the system to rollback to its previous state after a network or system failure or any types of event related to a particular connection handled by the Event Handler.*

***Risk:** *High. The system loads the particular events from the repository back to the UCM Manager.*

Constraints:

- **Usability:** System users are other subsystems(NCB and SE), which communicate through well-defined scripts and are easy to operate for other systems..
- **Reliability:** 15% failure rate is allowed for every 24hrs use.
- **Performance:** Requests should be handled in less than 2 minutes, if no other requests exist. Response time can be more depending on the number of events required to be loaded.
- **Supportability:** The command must be properly supported by Event handler of UCM.
- **Implementation:** Must be implemented in Java.

Modification History: - v1.2

***Owner:** Abhishek Bhattacharya

***Initiation date:** 09/10/2007

***Date last modified:** 09/23/2007

Use Case Save Event State

***Use Case ID:** *UCM_23 – Save Event*

Use Case Level: *Functional sub-use case.*

***Scenario:** *Synthesis Engine sends a control script to UCM containing a command to save an event 'E' for the sessionID 's1' with the active connection 'c1'.*

- **Actor:** *Synthesis Engine (SE), Network Communication Broker (NCB).*
- **Pre-conditions:**

1. *Participants are logged in and there is an active connection present.*

2. *A session with sessionID is generated and currently running in process.*

3. *The script received form the interpreter is in the legal format.*

4. *An exception is thrown from the NCB or the UCM itself in response to a set of particular events and the event state required to be saved.*

5. *The events are queued up and are ready to be saved in the repository.*

- **Description:**

  Trigger: *Use case begins when the Synthesis Engine requests the event to be saved via a control script.*

  The system responds by:

  1. *The logged event is saved in the Event Queue of the UCM.*
  2. *UCM obtains the session ID from the connection.*
  3. *UCM calls the Save Event function whenever the events are to be saved.*

- **Relevant requirements:** *{In this section reference is made to any other requirements documents such as industry standards or government regulations.}*
- **Post-conditions:** *The Event handler saves the corresponding event state into the Local Repository.*

***Alternative Courses of Action** *No alternate action.*

Extensions: *No Extensions.*

***Exceptions:**

1. *Unable to save the event state into the Local Repository.*
2. *Not able to communicate with the NCB.*

Concurrent Uses: *None*

***Related Use Cases:** *UCM_21 – Create Event.*

Decision Support

***Frequency:** *This use case will be performed every time a call is made to save a particular system event. It will be saved at least once for every session but depends on the size of the event queue.*

***Criticality:** High. *This important for the system to save a particular event and then loading it back whenever required.*

***Risk:** *High. The system saves the particular events from the event queue to the repository.*

Constraints:

- **Usability:** System users are other subsystems(NCB and SE), which communicate through well-defined scripts and are easy to operate for other systems.

- **Reliability:** 20% failure rate is allowed for every 24hrs use.
- **Performance:** Requests should be handled in less than 3 minutes, if no other requests exist. Response time can be more depending on the number of events required to be saved.
- **Supportability:** The command must be properly supported by the Event handler of UCM.
- **Implementation:** Must be implemented in Java.

Modification History: - v1.2

***Owner:** Abhishek Bhattacharya

***Initiation date:** 09/10/2007

***Date last modified:** 09/23/2007

Use Case Enable Media Initiator

***Use Case ID:** *UCM_24 – Enable Media Initiator*

Use Case Level: *Functional sub-use case.*

***Scenario:** *UCM receives a control script from the Synthesis Engine containing a command to add a media with name "MediaName1" and the medium URL as" www.medianame1.com"to the active connection "c1".*

Actor: *Synthesis Engine (SE), Network Communication Broker (NCB).*

- **Pre-conditions:**
    1. *The system has successfully created a connection.*
    2. *The initiator has completed the schema negotiation and is ready to send the media stream.*
- **Description:**

  Trigger: *Use case begins when the Synthesis Engine request the enabling of media via a control script.*

  The system responds by:

    1. *UCM obtains the session ID from the connection.*
    2. *Connection ID specifies the connection for which the media is initiated.*
    3. *This will enable the specified list of media to the designated connectionID.*

- **Relevant requirements:** *{In this section reference is made to any other requirements documents such as industry standards or government regulations.}*
- **Post-conditions:** *The audio connection is initiated by the sender.*

***Alternative Courses of Action** *No alternate action.*

Extensions: *No Extensions.*

***Exceptions:**

1. *No sessions can be found.*
2. *No media is available for communication.*

Concurrent Uses: *None*

Related Use Cases:UCM_01 -- Login, UCM_05 – AddParticipant, UCM_03 – Create Connection, UCM_14 – Send Schema.*

Decision Support

***Frequency:** *This use case will be performed at least once per connection instance. This will occur every time that a request from the Synthesis Engine is made to initiate the media after the schema negotiation is successful.*

***Criticality:** *High. This is required for anytime that the system requests the initiation for media communication. This is an important use case as without this there will be no media available for communication.*

***Risk:** *High. This use case is important for communication among users.*

Constraints:

- **Usability:** System users are other subsystems(NCB and SE), which communicate through well-defined scripts and are easy to operate for other systems.
- **Reliability:** 5% failure rate is allowed for every 24hrs use.
- **Performance:** Requests should be handled in less than 2 minutes, if no other requests exist. Response time can be more depending on the number of participants in the active connection.
- **Supportability:** The command must be properly supported by NCB and the Media Handler of UCM.
- **Implementation:** Must be implemented in Java.

Modification History: - v1.2

***Owner:** Abhishek Bhattacharya

***Initiation date:** 09/10/2007

***Date last modified:** 09/23/2007

Load Undefined Macro(misuse)

***Use Case ID:** *UCM_MU_LoadUnDefMacro*

Use Case Level: *Mis sub-use case*

***Scenario:** *A control script containing "remove participant" is passed down from the synthesis engine through the UCM-SE interface, to be executed. However, the script interpreter cannot find the corresponding macro for it. The system should not crash, but should provide nice error messages to the upper layers.*

- **Actor:** *Synthesis Engine (SE),*
- **Pre-conditions:**
    1. *The control script which has no macro definitions is passed down from the synthesis engine*
- **Description:**

  Trigger: *Use case begins when a control script from the synthesis engine is passed down to UCM for execution*

  The system responds by

    1. *UCM checks the type of the control script to be executed*
    2. *When UCM load the corresponding macro of the control script into memory, it can not find it in the local repository*
- **Relevant requirements:** *None*
- **Post-conditions:** *The system should not crash due to lack of macro definition.*

***Alternative Courses of Action** *No alternate action.*

Extensions: *None*

Concurrent Uses: *None*

***Related Use Cases:**

Decision Support

***Frequency:** An average of 4 times per application use.

***Criticality:** High

***Risk:** *Low, this use case is performed by a layer that is isolated from any system resource.*

Constraints:

- **Usability:** This use case should provide nice error messages to the user..
- **Performance:** The total process for searching the macro definition should not be more than 5 seconds, before the user gets bored.
- **Implementation:** Must be implemented in Java.
- **Operation:** The running UCM is invoked by SE and supported by NCB.

Modification History – v1.2

***Owner:** Guangqiang Zhao

***Initiation date:** 09/08/2007

***Date last modified:** 09/21/2007

Respond to Undefined Macro (security)

***Use Case ID:** *UCM_SU_LoadUnDefMacro*

Use Case Level: *Mis sub-use case*

***Scenario:** *A control script containing "sendDemandForm" is passed down from the synthesis engine through the UCM-SE interface. The macro for "sendDemandForm" is not defined in the local repository. The system notifies to SE that this command is not defined and suggested a related one that is already defined "sendForm".*

- **Actor:** *Synthesis Engine (SE),*
- **Pre-conditions:**
  1. *The control script which has no macro definitions is passed down from the synthesis engine*
- **Description:**

  Trigger: *Use case begins when a undefined control script from the synthesis engine is passed down to UCM for execution*

  The system responds by

1. *Respond to Synthesis Engine that this command is undefined in the repository*
2. *Look for a related macro for this command and recommend to the synthesis engine*

- **Relevant requirements:** *None*
- **Post-conditions:** *Possible attack to the UCM is prevented.*

***Alternative Courses of Action** *No alternate action.*

Extensions: *None*

Concurrent Uses: *None*

***Related Use Cases:** the loadUnDefMacro use case

Decision Support

***Frequency:** Not frequent, but might happen once or twice per application use..

***Criticality:** High

***Risk:** *Low, this use case is performed by a layer that is isolated from any system resource.*

Constraints:

- **Usability:** This use case should provide nice error messages to the user..
- **Performance:** The responses should be returned in at most 2 seconds
- **Implementation:** Must be implemented in Java.
- **Operation:** The running UCM is invoked by SE and supported by NCB.

Modification History –v1.2

***Owner:** Guangqiang Zhao

***Initiation date:** 09/08/2007

***Date last modified:** 09/21/2007

Use Case Save Macro Instance(misuse)

Use Case ID: *UCM_MU_SaveMacroInstane*

Use Case Level:

***Scenario:** UCM has requested to save the instance for Macro login with user name "Burke" and pw = "1234". UCM Manager then saves that macro instance into the local repository. UCM manager also maps the login script with the macro instance provided.*

- **Actor:** *Macro developer, Local repository*
- **Pre-conditions:** *The macro definition is already defined and ready for storage.*
- **Description:**

  Trigger: *The user initiates an action requesting to save the macro instance*

  The system responds by:

  1. *Saving the macro instance into the local repository as a string or a text file*
  2. *When recording the script-to-macro mapping, UCM finds the macro has existed in the local repository.*

- **Relevant requirements:** *None*
- **Post-conditions:**

  1. *The system should not overwrite the previous one without notification*
  2. *The system should not crash due to this unexpected event.*

***Alternative Courses of Action** : No alternative actions*

Extensions: *None*

***Exceptions:** None*

1. *The macro instance is not successfully saved into the local repository, this exceptions has to be handled by the Exception Handler.*

Concurrent Uses: *None*

***Related Use Cases:** the SaveExistedMacro security use case*

Decision Support

***Frequency:** This use case does not happen frequently, at most once or twice per application use..*

***Criticality:** Low, not core functionality*

***Risk:** Low. A layer that is isolated from any system resource performs this use case.*

Constraints:

- **Supportability:** This use case should be able to save any form of macro instances.
- **Interface:** This macro should only have one interface: saveMacro() to the macro developer.
- **Implementation:** This use case should be able to save the macro in all the mainstream file systems.
- **Performance:** This use case should be handled within 10 milliseconds.
- **Implementation:** Must be implemented in Java.

Modification History – *v1.2*

***Owner:** Guangqiang Zhao

***Initiation date:** 09/08/2007

***Date last modified:** 09/21/2007

Respond to Saving Existed Macro(security)

Use Case ID: *UCM_SU_SaveExistedMacro*

Use Case Level:

***Scenario:** *UCM has requested to save the instance for Macro login with user name "Burke" and pw = "1234", which is already existing in the repository. The system should notify the user of this accident and prompt the user either to disregard the current macro or to overwrite the previous one.*

- **Actor:** *Macro developer, Local repository*
- **Pre-conditions:** *The macro definition is already defined and ready for storage.*
- **Description:**

  Trigger: *The user initiates an action requesting to save an existed macro instance*

  The system responds by:

1. *Notifying the user that the macro instance has already existed*
2. *Provide the use with two solutions: either disregard the current macro instance or to overwrite the previous one.*

- **Relevant requirements:** *None*

- **Post-conditions:** Either *the current macro is saved into the local repository or the previous one remains.*

***Alternative Courses of Action** *: No alternative actions*

Extensions: *None*

***Exceptions:** *None*

Concurrent Uses: *None*

***Related Use Cases:** *the SaveMacroInstane misuse case*

Decision Support

***Frequency:** *This use case does not happen frequently, at most once or twice per application use..*

***Criticality:** *Low, not core functionality*

***Risk:** *Low. A layer that is isolated from any system resource performs this use case.*

Constraints:

- **Supportability:** This use case should be able to save any form of macro instances.
- **Interface:** This macro should only have one interface: saveMacro() to the macro developer.
- **Implementation:** This use case should be able to save the macro in all the mainstream file systems.
- **Performance:** This use case should be handled within 10 milliseconds
- **Implementation:** Must be implemented in Java.

Modification History – *v1.2*

***Owner:** Guangqiang Zhao

***Initiation date:** 09/08/2007

***Date last modified:** 09/21/2007

10.3 Appendix C – User Interface designs.

cvm

Class CVM

```
java.lang.Object
  └cvm.CVM
```

All Implemented Interfaces:

cvm.session.media.sip.event.CommunicationsListener, cvm.session.media.sip.simple.event.ContactListChangeListener, java.util.EventListener, cvm.ICVM, cvm.session.media.sip.security.SecurityAuthority, cvm.session.media.sip.simple.SubscriptionAuthority

---

public class **CVM**
extends java.lang.Object
implements cvm.session.media.sip.security.SecurityAuthority, cvm.session.media.sip.simple.SubscriptionAuthority, cvm.session.media.sip.event.CommunicationsListener, cvm.session.media.sip.simple.event.ContactListChangeListener, cvm.ICVM

Author:

Field Summary

`cvm.session.party.PartyGroup`	**`partyList`**

Constructor Summary

`CVM``()`
`CVM``(executionengine.ExecutionEngine ee)`

Method Summary	
boolean	**addedMedia**(java.lang.String sid, java.lang.String mediaType)
boolean	**addedParty**(int sid, java.lang.String userName)
boolean	**addMedia**(java.lang.String sid, java.lang.String media_type, java.lang.String media_location) related to media tranmission
boolean	**addParty**(java.lang.String sid, java.lang.String name)
boolean	**addPartyToPartyList**(java.lang.String displayName, java.lang.String presenceUri)
int	**adjustMediaQoS**(java.lang.String sid, java.lang.String media_type, java.lang.String media_location, int quality)
void	**callReceived**(cvm.session.media.sip.event.CallEvent evt)
void	**callRejectedLocally**(cvm.session.media.sip.event.CallRejectedEvent evt)
void	**callRejectedRemotely**(cvm.session.media.sip.event.CallRejectedEvent evt)
void	**communicationsErrorOccurred**(cvm.session.media.sip.event.CommunicationsErrorEvent evt)
void	**contactAdded**(cvm.session.media.sip.simple.event.ContactAddedEvent evt)
void	**contactRemoved**(cvm.session.media.sip.simple.event.ContactRemovedEvent evt)
boolean	**createdSession**(int sid)
void	**createSession**(java.lang.String uuid)
boolean	**destroySession**(java.lang.String sid)

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

partyList

public cvm.session.party.PartyGroup **partyList**

Constructor Detail

CVM

public **CVM**()

CVM

public **CVM**(executionengine.ExecutionEngine ee)

Method Detail

notificationReceived

public void **notificationReceived**(cvm.session.media.sip.simple.event.NotificationReceivedEvent evt)

Specified by:

notificationReceived in interface cvm.session.media.sip.simple.event.ContactListChangeListener

---

contactAdded

public void **contactAdded**(cvm.session.media.sip.simple.event.ContactAddedEvent evt)

Specified by:

contactAdded in interface cvm.session.media.sip.simple.event.ContactListChangeListener

---

contactRemoved

public void **contactRemoved**(cvm.session.media.sip.simple.event.ContactRemovedEvent evt)

Specified by:

contactRemoved in interface cvm.session.media.sip.simple.event.ContactListChangeListener

---

callReceived

public void **callReceived**(cvm.session.media.sip.event.CallEvent evt)

Specified by:

callReceived in interface cvm.session.media.sip.event.CommunicationsListener

---

messageReceived

```
public void
messageReceived(cvm.session.media.sip.event.MessageEvent evt)
```

Specified by:

`messageReceived` in interface `cvm.session.media.sip.event.CommunicationsListener`

---

callRejectedLocally

```
public void
callRejectedLocally(cvm.session.media.sip.event.CallRejectedEvent evt)
```

Specified by:

`callRejectedLocally` in interface `cvm.session.media.sip.event.CommunicationsListener`

---

callRejectedRemotely

```
public void
callRejectedRemotely(cvm.session.media.sip.event.CallRejectedEvent evt)
```

Specified by:

`callRejectedRemotely` in interface `cvm.session.media.sip.event.CommunicationsListener`

---

registered

```
public void
registered(cvm.session.media.sip.event.RegistrationEvent evt)
```

Specified by:

`registered` in interface `cvm.session.media.sip.event.CommunicationsListener`

---

registering

```
public void
registering(cvm.session.media.sip.event.RegistrationEvent evt)
```

Specified by:

`registering` in interface `cvm.session.media.sip.event.CommunicationsListener`

---

unregistered

```
public void
unregistered(cvm.session.media.sip.event.RegistrationEvent evt)
```

Specified by:

`unregistered` in interface `cvm.session.media.sip.event.CommunicationsListener`

---

unregistering

```
public void
unregistering(cvm.session.media.sip.event.RegistrationEvent evt)
```

Specified by:

`unregistering` in interface `cvm.session.media.sip.event.CommunicationsListener`

receivedUnknownMessage

```
public void
receivedUnknownMessage(cvm.session.media.sip.event.UnknownMessageEvent
evt)
```

Specified by:

receivedUnknownMessage in interface cvm.session.media.sip.event.CommunicationsListener

communicationsErrorOccurred

```
public void
communicationsErrorOccurred(cvm.session.media.sip.event.CommunicationsE
rrorEvent evt)
```

Specified by:

communicationsErrorOccurred in interface cvm.session.media.sip.event.CommunicationsListener

register

```
public boolean register(java.lang.String uri,
                        java.lang.String displayName,
                        java.lang.String passwd)
```

Registers a user with the sip server. Note: Requires the uri to be in the format: "sip:"+username+"@bobcat.cs.fiu.edu".

Specified by:

register in interface cvm.ICVM

obtainCredentials

```
public cvm.session.media.sip.security.UserCredentials
obtainCredentials(java.lang.String realm,

cvm.session.media.sip.security.UserCredentials defaultValues)
```

Implements obtainCredentials from SecurityAuthority.

Specified by:

obtainCredentials in interface cvm.session.media.sip.security.SecurityAuthority

Parameters:

realm - the realm that credentials are needed for

defaultValues - the values to propose the user by default

Returns:

the credentials for the specified realm or null if no credentials could be obtained

requestSubscriptionAuthorization

```
public cvm.session.media.sip.simple.SubscriptionAuthorizationResponse
requestSubscriptionAuthorization(java.lang.String displayName,

java.lang.String address,

java.lang.String message,
```

java.lang.String[] acceptedResponses)

Description copied from interface: cvm.session.media.sip.simple.SubscriptionAuthority

this method would (indirectly) ask the user for their consent or a set of preauthorized subscribers. The method operates in a blocking manner. It is therefore recommended that prior to calling it, the PresenceAgent should send an ACCEPTED response and send a NOTIFY containing bogus presence information data (such data may be obtained by calling the getPresenceInformationData method with a null authorization parameter).

Specified by:

requestSubscriptionAuthorization in interface cvm.session.media.sip.simple.SubscriptionAuthority

---

launch

public void **launch**()

---

shutdown

public void **shutdown**()

Specified by:

shutdown in interface cvm.ICVM

---

getUserName

public java.lang.String **getUserName**()

---

main

public static void **main**(java.lang.String[] args)

---

createSession

public void **createSession**(java.lang.String uuid)

Specified by:

createSession in interface cvm.ICVM

---

joinSession

public void **joinSession**(java.lang.String userid,
java.lang.String sid)

Specified by:

joinSession in interface cvm.ICVM

---

destroySession

public boolean **destroySession**(java.lang.String sid)

Specified by:

destroySession in interface cvm.ICVM

---

getCapability

public boolean **getCapability**(java.lang.String user,

```
                                java.lang.String media_type)
```

related to party

Specified by:

`getCapability` in interface `cvm.ICVM`

---

getCapabilities

```
public java.util.ArrayList getCapabilities()
```

Specified by:

`getCapabilities` in interface `cvm.ICVM`

---

addPartyToPartyList

```
public boolean addPartyToPartyList(java.lang.String displayName,
                                   java.lang.String presenceUri)
```

---

addParty

```
public boolean addParty(java.lang.String sid,
                        java.lang.String name)
```

Specified by:

`addParty` in interface `cvm.ICVM`

---

removeParty

```
public boolean removeParty(java.lang.String sid,
                           java.lang.String name)
```

Specified by:

`removeParty` in interface `cvm.ICVM`

---

addMedia

```
public boolean addMedia(java.lang.String sid,
                        java.lang.String media_type,
                        java.lang.String media_location)
```

related to media tranmission

Specified by:

`addMedia` in interface `cvm.ICVM`

---

removeMedia

```
public boolean removeMedia(java.lang.String sid,
                           java.lang.String media_type,
                           java.lang.String media_location)
```

Specified by:

`removeMedia` in interface `cvm.ICVM`

---

sendMedia

```
public boolean sendMedia(java.lang.String sid,
                         java.lang.String media_type,
                         java.lang.String media_location)
```

Specified by:

`sendMedia` in interface `cvm.ICVM`

stopMedia

```
public boolean stopMedia(java.lang.String sid,
                         java.lang.String media_type,
                         java.lang.String media_location)
```

Specified by:

`stopMedia` in interface `cvm.ICVM`

---

adjustMediaQoS

```
public int adjustMediaQoS(java.lang.String sid,
                          java.lang.String media_type,
                          java.lang.String media_location,
                          int quality)
```

Specified by:

`adjustMediaQoS` in interface `cvm.ICVM`

---

remoteSessionOpen

```
public boolean remoteSessionOpen(int sid)
```

related to callbacks from Execution Engine

---

remotePartyAdded

```
public boolean remotePartyAdded(int sid,
                                java.lang.String user)
```

---

mediaRequest

```
public boolean mediaRequest(java.lang.String userName,
                            java.lang.String media_type,
                            java.lang.String media_location)
```

---

createdSession

```
public boolean createdSession(int sid)
```

---

addedParty

```
public boolean addedParty(int sid,
                          java.lang.String userName)
```

---

addedMedia

```
public boolean addedMedia(java.lang.String sid,
                          java.lang.String mediaType)
```

---

sendSchema

```
public boolean sendSchema(java.lang.String sid,
                          java.lang.String userName,
                          java.lang.String schema)
```

Specified by:

`sendSchema` in interface `cvm.ICVM`

---

sessionNegotiation

```
public boolean sessionNegotiation(java.lang.String sid,
                                  java.lang.String userName,
                                  java.lang.String sessionMsg)
```

---

snCreateSession

```
public boolean snCreateSession(java.lang.String sid,
                               java.lang.String userName)
```

---

getNetworkAddressManager

```
public cvm.sc.impl.netaddr.NetworkAddressManagerServiceImpl
getNetworkAddressManager()
```

10.4 Appendix D – Detailed Class Diagrams

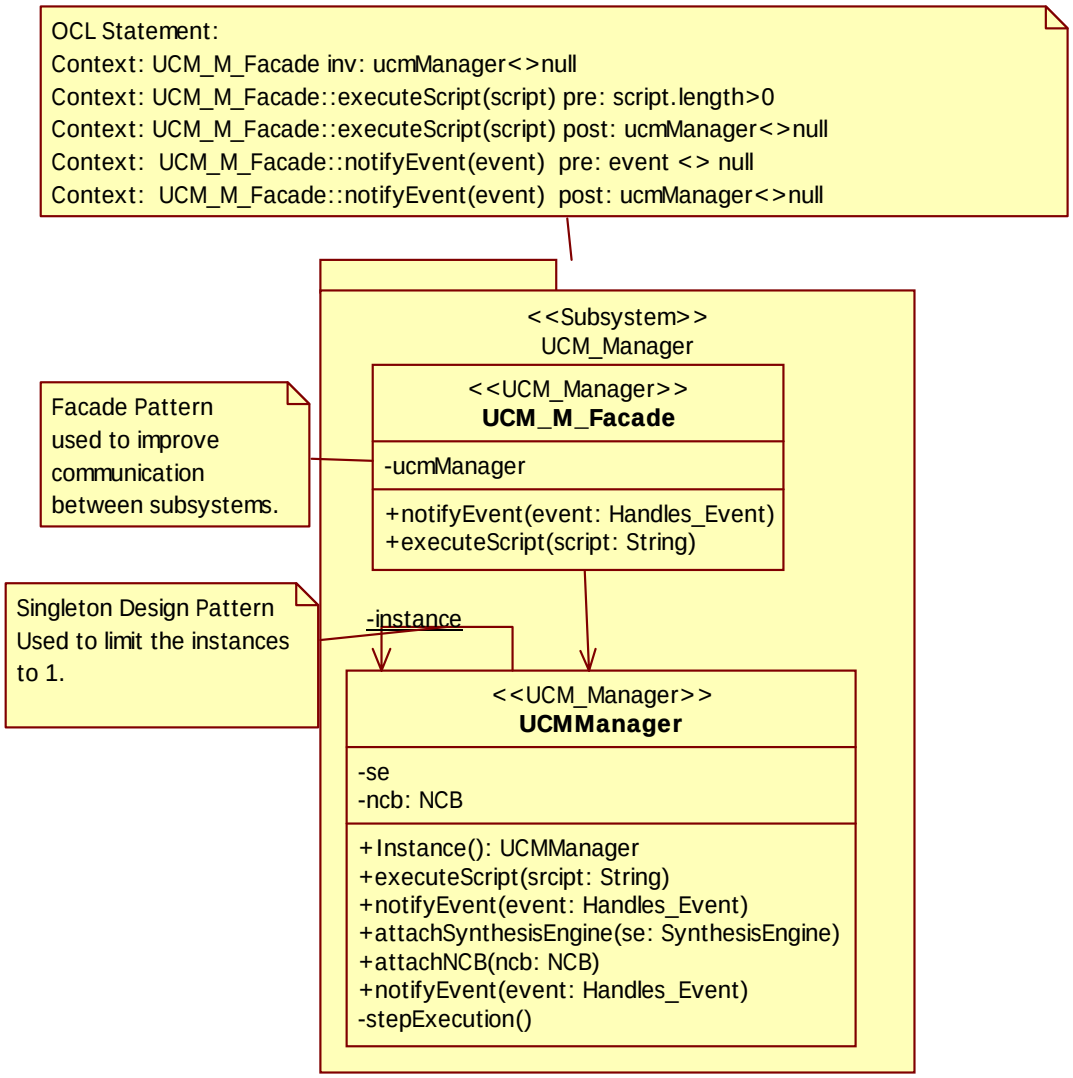


Fig C.1 UCM_Manager class diagram.

Description: The UCM_Manager controls the work flow inside the UCM system. This class also notifies the overlying system of any event that are specific to it.

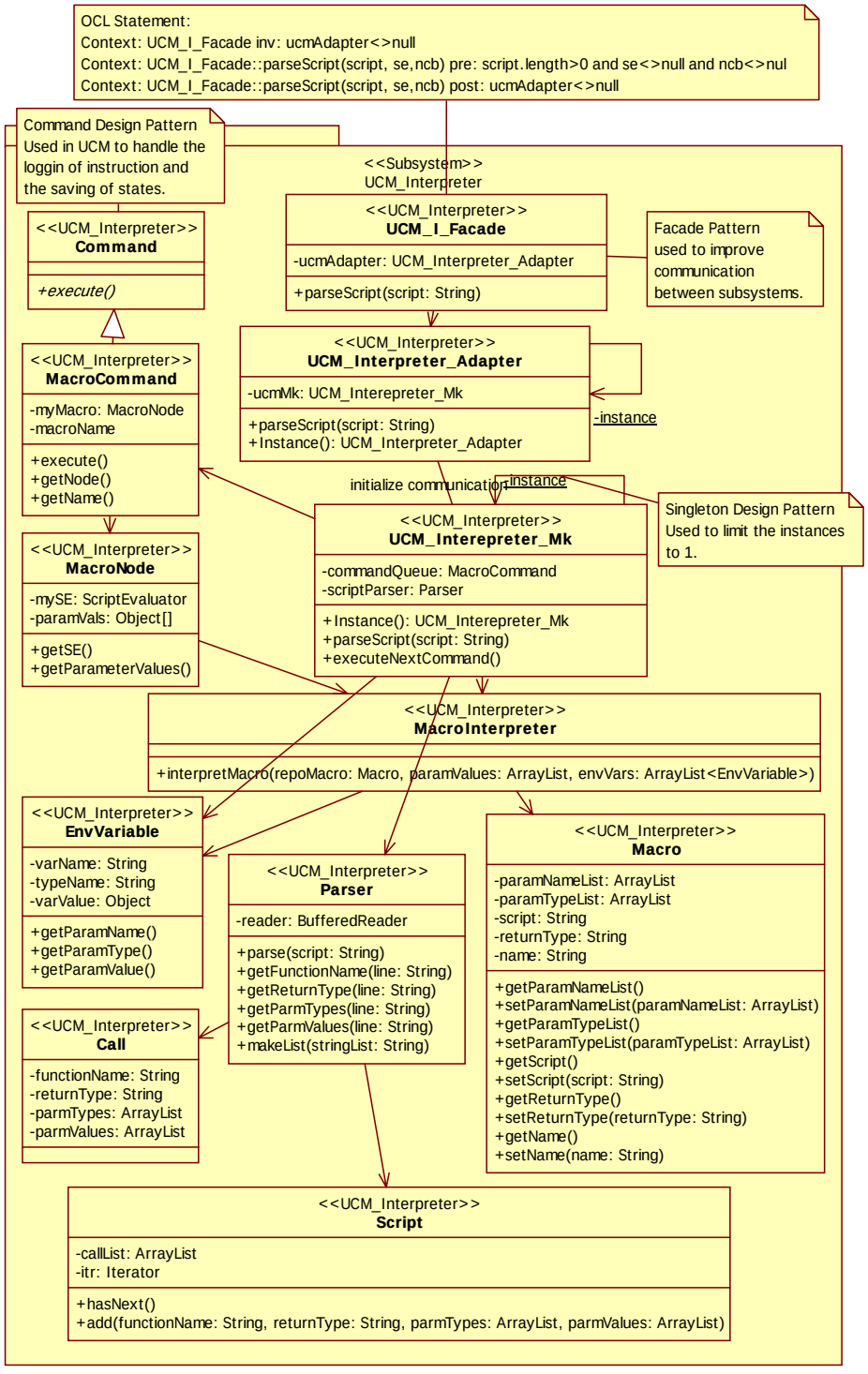


Fig C.2 UCM_Interpreter detailed class diagram.

Description: The UCM_Interpreter handels the pasrsing and execution of a control script. Every script is parsed into a script object containing a list of call object for every function inside the original control script. Every call is then converted into a command for execution.

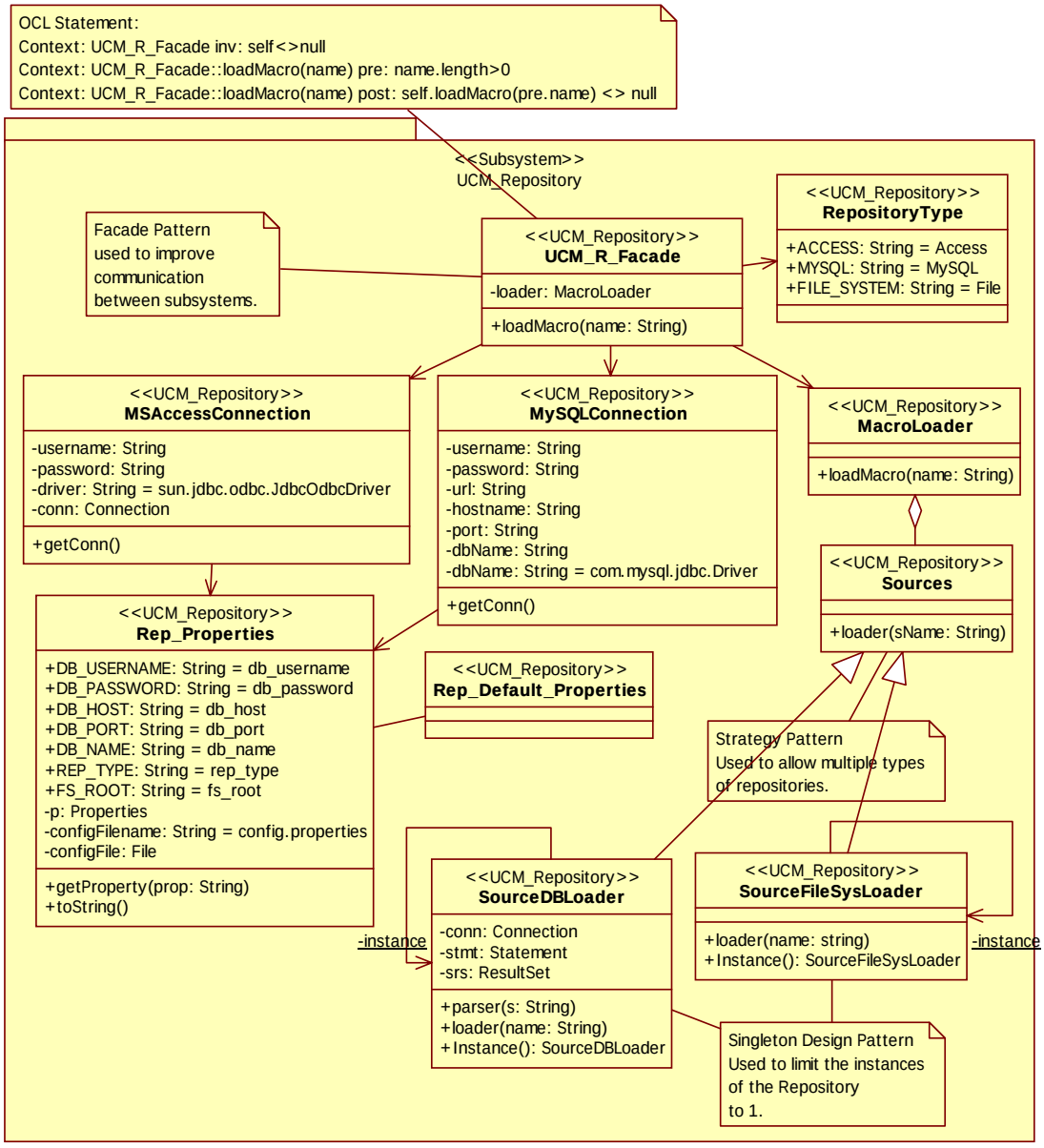


Fig C.3 UCM_Repository detailed class diagram.

Description: The UCM_Repository stores the macros for the execution of the control scripts. Macros can be added at any time. This ensures the extensibility of the application without having to write or change any core code.

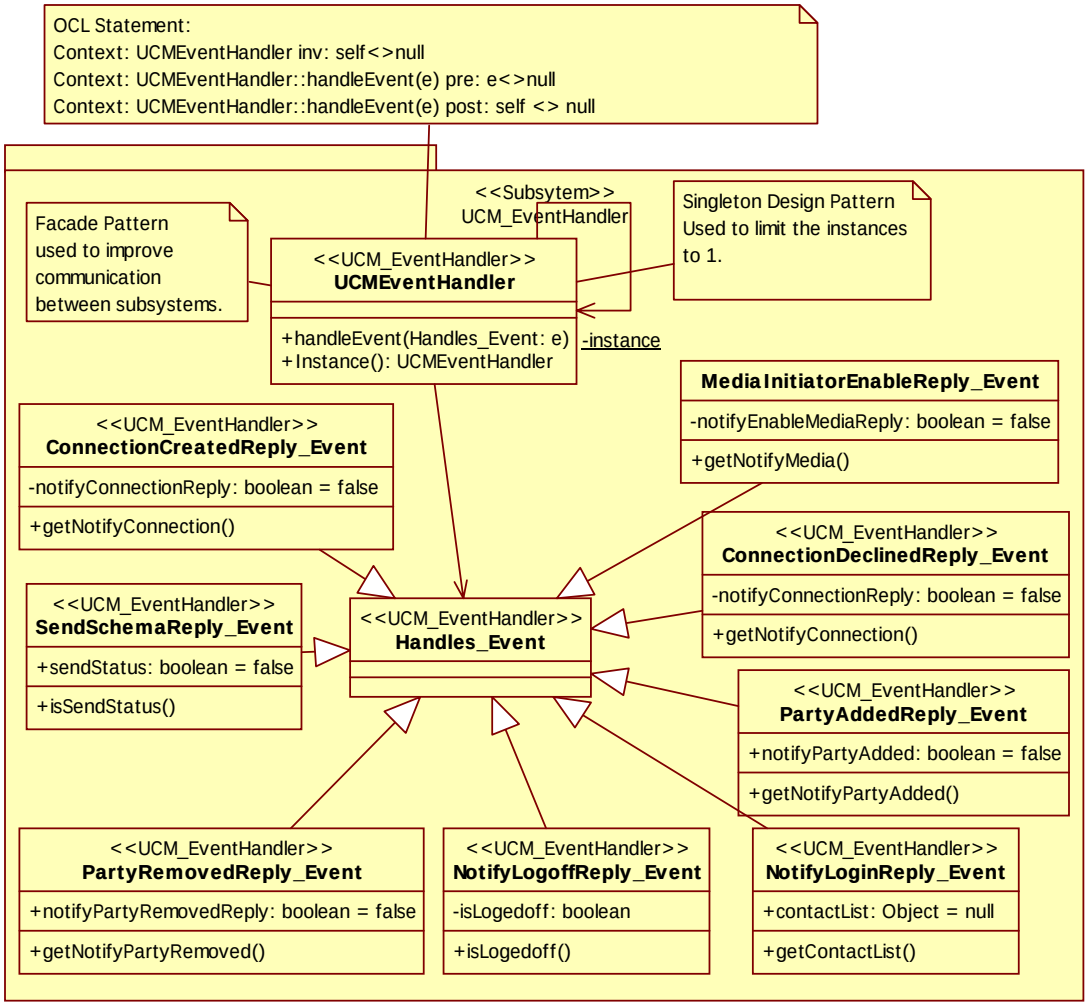


Fig C.4 UCM_EventHandler detailed class diagram.

Description: The UCM_EventHandler will coordinate and orchestrate the events raised by other subsystems as well as deciding what to do in each case.

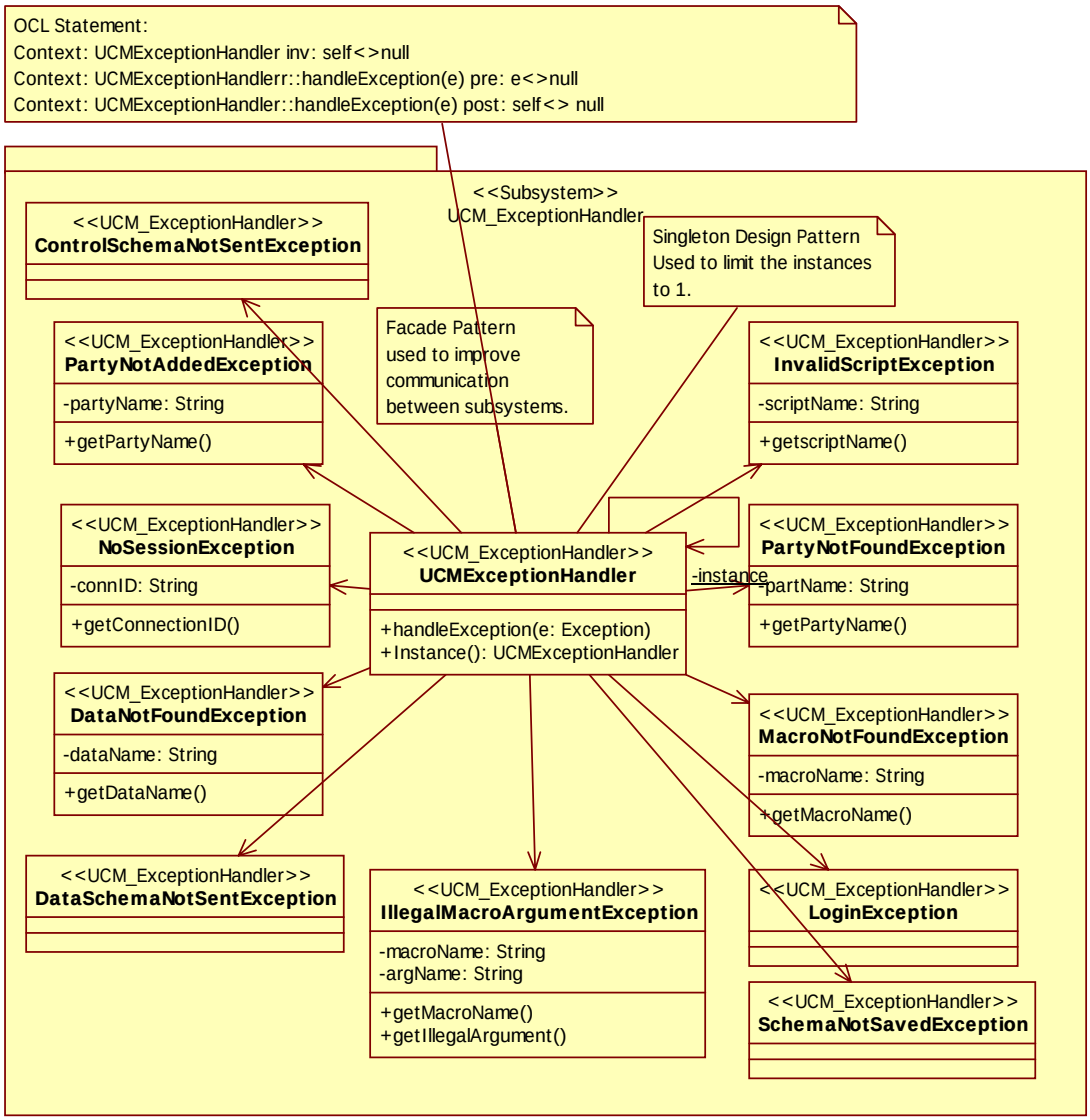


Fig C.5 UCM_ExcpetionHandler detailed class diagram.

Description: - The UCM_ExceptionHandler will be responsible for deciding how to act on exceptions received due to control script faults, NCB specific messages, or bad function call returns.

10.5 Appendix E – Class Interfaces

cvm.ucm.handlers
Class UCMEventHandler

```
java.lang.Object
  └ cvm.ucm.handlers.UCMEventHandler
```

All Implemented Interfaces:

Uses_Listener, java.util.EventListener

```
public class UCMEventHandler
extends java.lang.Object
implements Uses_Listener
```

Field Summary

private static UCMEventHandler	**instance**

Constructor Summary

private	**UCMEventHandler**() The UCMEventhandler constructor register itself with the event source, namely the NCBEventObjectManager, which will fire the events.

Method Summary

private void	**handleEvent**(Handles_Event e) This method will handle all the events coming from NCB OCL Statement: Context: UCMEventHandler::handleEvent(e) pre: e<>null Context: UCMEventHandler::handleEvent(e) post: self <> null
static UCMEventHandler	**Instance**() Implementation of the instance method as in the Singleton design pattern.
void	**use**(Handles_Event event)

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

instance
`private static UCMEventHandler instance`

Constructor Detail

UCMEventHandler
`private UCMEventHandler()`

The UCMEventhandler constructor register itself with the event source, namely the NCBEventObjectManager, which will fire the events.

Method Detail

Instance
`public static UCMEventHandler Instance()`

Implementation of the instance method as in the Singleton design pattern.

Returns:

UCMEventhandler instnace

---

handleEvent
`private void handleEvent(Handles_Event e)`

This method will handle all the events coming from NCB OCL Statement: Context: UCMEventHandler::handleEvent(e) pre: e<>null Context: UCMEventHandler::handleEvent(e) post: self <> null

Parameters:

`e` - the event that will be handled by the handler class

---

use
`public void use(Handles_Event event)`

Specified by:

`use` in interface `Uses_Listener`

---

cvm.ucm.handlers

Class UCMEventObjectManager

```
java.lang.Object
  └ cvm.ucm.handlers.UCMEventObjectManager
```

`public class UCMEventObjectManager`
`extends java.lang.Object`

This class handles the firing of events for UCM.

Author:

Frank Hernandez

Field Summary

private static UCMEventObjectManager	**instance**
private static java.util.Vector	**listeners**
private static java.lang.Object	**syncObject_**

Constructor Summary

private	**UCMEventObjectManager**()

Method Summary

void	**addDownListener**(java.util.EventListener listener)
void	**addUpListener**(Uses_Listener listener)
private void	**fireUpEventUCM**(Handles_Event event) General Event Triger.
static UCMEventObjectManager	**Istance**() Implementation of the Instance method as in the singleton design pattern.
void	**notifyConnectionCreatedReply_Event**(boolean notifyConnectionReply) This methods notifies UCM of a ConnectionCreatedReply.
void	**notifyConnectionDeclinedReply_Event**(boolean notifyConnectionReply) This methods notifies UCM of a ConnectionDeclineReply.
void	**notifyControlSchemaNotSentException_Event**() Notify SE of a ControlSchemaNotSentException
void	**notifyDataSchemaNotSentException_Event**() Notify SE of a DataSchemaNotSentException
void	**notifyLoginExceptionEvent**() Notify SE of a loginException
void	**notifyLoginReply**(java.lang.Object contactlist)

	This method will notify UCM of a LoginReply.
void	**notifyLogoffReply**(boolean success) This method will notify UCM of a LogoffReply.
void	**notifyMediaInitiatorEnableReply_Event**(boolean partyRemoved) This methods notifies UCM of a MediaInitiatorEnableReply.
void	**notifyNoSessionException_Event**(java.lang.String sID) Notify SE of a noSessionException
void	**notifyPartyAddedReply**(boolean partyAdded) This methods notifies UCM of a PartyAddedReply.
void	**notifyPartyNotAddedException_Event**(java.lang.String userID) Notify SE of a PartyNotAddedException
void	**notifyPartyRemovedReply_Event**(boolean partyRemoved) This methods notifies UCM of a PartyRemovedReply.
void	**notifySchemaNotSavedException_Event**() Notify SE of SchemaNotSavedException
void	**notifySendSchemaReply_Event**(boolean sendStatus) This method notifes UCM of a SendSchemaReply.
void	**notifyUnavailableMedia_Event**() Notify SE of UnavailableMedia
void	**notifyUnrecognizedEvent**()
void	**notifyUserProfileCreatedEvent**(UserProfile usrProfile) Event notification function.
void	**removeUpListener**(Uses_Listener listener)

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

listeners
```
private static java.util.Vector listeners
```

---

instance
```
private static UCMEventObjectManager instance
```

---

syncObject_
```
private static java.lang.Object syncObject_
```

Constructor Detail

UCMEventObjectManager
```
private UCMEventObjectManager()
```

Method Detail

Istance
```
public static UCMEventObjectManager Istance()
```
Implementation of the Instance method as in the singleton design pattern.

Returns:

---

addUpListener
```
public void addUpListener(Uses_Listener listener)
```

---

addDownListener
```
public void addDownListener(java.util.EventListener listener)
```

---

removeUpListener
```
public void removeUpListener(Uses_Listener listener)
```

---

notifyLoginReply
```
public void notifyLoginReply(java.lang.Object contactlist)
```
This method will notify UCM of a LoginReply.

Parameters:

`contactlist` -

---

notifyLogoffReply
```
public void notifyLogoffReply(boolean success)
```
This method will notify UCM of a LogoffReply.

Parameters:

`contactlist` -

---

notifyPartyAddedReply
```
public void notifyPartyAddedReply(boolean partyAdded)
```
This methods notifies UCM of a PartyAddedReply.

Parameters:

`partyAdded` -

---

notifyPartyRemovedReply_Event

`public void notifyPartyRemovedReply_Event(boolean partyRemoved)`

This methods notifies UCM of a PartyRemovedReply.

Parameters:

`partyRemoved` -

---

notifyMediaInitiatorEnableReply_Event

`public void notifyMediaInitiatorEnableReply_Event(boolean partyRemoved)`

This methods notifies UCM of a MediaInitiatorEnableReply.

Parameters:

`partyRemoved` -

---

notifySendSchemaReply_Event

`public void notifySendSchemaReply_Event(boolean sendStatus)`

This method notifes UCM of a SendSchemaReply.

Parameters:

`sendStatus` -

---

notifyConnectionCreatedReply_Event

`public void notifyConnectionCreatedReply_Event(boolean notifyConnectionReply)`

This methods notifies UCM of a ConnectionCreatedReply.

Parameters:

`notifyConnectionReply` -

---

notifyConnectionDeclinedReply_Event

`public void notifyConnectionDeclinedReply_Event(boolean notifyConnectionReply)`

This methods notifies UCM of a ConnectionDeclineReply.

Parameters:

`notifyConnectionReply` -

---

notifyUserProfileCreatedEvent

`public void notifyUserProfileCreatedEvent(UserProfile usrProfile)`

Event notification function. This function wil fire the event to notify the SE.

Parameters:

`usrProfile` -

---

notifyLoginExceptionEvent

`public void notifyLoginExceptionEvent()`

Notify SE of a loginException

---

notifyNoSessionException_Event

```
public void notifyNoSessionException_Event(java.lang.String sID)
```

Notify SE of a noSessionException

---

notifyPartyNotAddedException_Event

```
public void notifyPartyNotAddedException_Event(java.lang.String userID)
```

Notify SE of a PartyNotAddedException

---

notifyDataSchemaNotSentException_Event

```
public void notifyDataSchemaNotSentException_Event()
```

Notify SE of a DataSchemaNotSentException

---

notifyControlSchemaNotSentException_Event

```
public void notifyControlSchemaNotSentException_Event()
```

Notify SE of a ControlSchemaNotSentException

---

notifyUnavailableMedia_Event

```
public void notifyUnavailableMedia_Event()
```

Notify SE of UnavailableMedia

---

notifySchemaNotSavedException_Event

```
public void notifySchemaNotSavedException_Event()
```

Notify SE of SchemaNotSavedException

---

notifyUnrecognizedEvent

```
public void notifyUnrecognizedEvent()
```

---

fireUpEventUCM

```
private void fireUpEventUCM(Handles_Event event)
```

General Event Triger. This method will fire the given event.

Parameters:

`event` -

---

cvm.ucm.handlers

Class UCMExceptionHandler

```
java.lang.Object
  └ cvm.ucm.handlers.UCMExceptionHandler
```

```
public class UCMExceptionHandler
extends java.lang.Object
```

Field Summary

private static UCMExceptionHandler	**instance**
private static UCMEventObjectManager	**ucmEManager**

Constructor Summary

private	**UCMExceptionHandler**()

Method Summary

void	**handleException**(java.lang.Exception e) This method will handle all the exception OCL Statement: Context: UCMExceptionHandlerr::handleException(e) pre: e<>null Context: UCMExceptionHandler::handleException(e) post: self<> null
static UCMExceptionHandler	**Instance**() Implementation of the instance method as in the Singleton design pattern.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

instance

private static UCMExceptionHandler **instance**

---

ucmEManager

private static UCMEventObjectManager **ucmEManager**

Constructor Detail

UCMExceptionHandler

private **UCMExceptionHandler**()

Method Detail

handleException

public void **handleException**(java.lang.Exception e)

This method will handle all the exception OCL Statement: Context: UCMExceptionHandlerr::handleException(e) pre: e<>null Context: UCMExceptionHandler::handleException(e) post: self<> null

Parameters:

e - the exception that will be handled by the handler class

Instance

```
public static UCMExceptionHandler Instance()
```

Implementation of the instance method as in the Singleton design pattern.

Returns:

UCMExceptionHandler instnace

---

cvm.ucm.handlers.event

Class ConnectionCreatedReply_Event

```
java.lang.Object
  └java.util.EventObject
      └cvm.ucm.handlers.event.Handles_Event
          └cvm.ucm.handlers.event.ConnectionCreatedReply_Event
```

All Implemented Interfaces:

java.io.Serializable

```
public class ConnectionCreatedReply_Event
extends Handles_Event
```

This class will represent an event coming from NCB that encapsulate the reply of the create connection request

Author:

Guangqiang Zhao

See Also:

Serialized Form

Field Summary

private boolean	**notifyConnectionReply** the variable holding whether the connection is created successfully or not

Fields inherited from class java.util.EventObject
source

Constructor Summary

`ConnectionCreatedReply_Event(java.lang.Object eventSource, boolean notifyConnectionReply)` The constructor

Method Summary

`boolean`	`getNotifyConnection()` This method returns whether the connection is created successfully or not

Methods inherited from class java.util.EventObject
`getSource, toString`

Methods inherited from class java.lang.Object
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Field Detail

notifyConnectionReply

```
private boolean notifyConnectionReply
```

the variable holding whether the connection is created successfully or not

Constructor Detail

ConnectionCreatedReply_Event

```
public ConnectionCreatedReply_Event(java.lang.Object eventSource,
                                    boolean notifyConnectionReply)
```

The constructor

Parameters:

`eventSource` - the object that fires this event

`notifyConnectionReply` - containing the result of the create connection request

Method Detail

getNotifyConnection

```
public boolean getNotifyConnection()
```

This method returns whether the connection is created successfully or not

Returns:

the result of the create connection request

---

cvm.ucm.handlers.event

Class ConnectionDeclinedReply_Event

`java.lang.Object`

└ java.util.EventObject
  └ cvm.ucm.handlers.event.Handles_Event
    └ **cvm.ucm.handlers.event.ConnectionDeclinedReply_Event**

All Implemented Interfaces:

java.io.Serializable

```
public class ConnectionDeclinedReply_Event
extends Handles_Event
```

This class will represent an event encapsulate the reply of the decline connection request

Author:

Frank Hernandez

See Also:

Serialized Form

Field Summary

private boolean	**notifyConnectionReply** The variable holds whether the connection is declined successfully or not

Fields inherited from class java.util.EventObject
source

Constructor Summary

ConnectionDeclinedReply_Event(java.lang.Object eventSource, boolean notifyConnectionReply) The constructor

Method Summary

boolean	**getNotifyConnection**() This method returns whether the connection is declined successfully or not

Methods inherited from class java.util.EventObject
getSource, toString

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

notifyConnectionReply

```
private boolean notifyConnectionReply
```

The variable holds whether the connection is declined successfully or not

Constructor Detail

ConnectionDeclinedReply_Event

```
public ConnectionDeclinedReply_Event(java.lang.Object eventSource,
                                     boolean notifyConnectionReply)
```

The constructor

Parameters:

`eventSource` - the object that fires this event

`notifyConnectionReply` - containing the result of the declined connection request

Method Detail

getNotifyConnection

```
public boolean getNotifyConnection()
```

This method returns whether the connection is declined successfully or not

Returns:

the result of the create connection request

---

cvm.ucm.handlers.event

Class Handles_Event

```
java.lang.Object
  └java.util.EventObject
      └cvm.ucm.handlers.event.Handles_Event
```

All Implemented Interfaces:

java.io.Serializable

Direct Known Subclasses:

ConnectionCreatedReply_Event, ConnectionDeclinedReply_Event, ControlSchemaNotSentException_Event, DataSchemaNotSentException_Event, LoginExcetption_Event, MediaInitiatorEnableReply_Event, NoSessionException_Event, NotifyLoginReply_Event, NotifyLogoffReply_Event, PartyAddedReply_Event, PartyNotAddedException_Event, PartyRemovedReply_Event, SchemaNotSavedException_Event, SendSchemaReply_Event, UnavailableMedia_Event, UserProfileCreatedEvent

---

```
public class Handles_Event
extends java.util.EventObject
```

This class is the parent class for all classes that represent an event coming up from a lower level to a higher level

See Also:

Serialized Form

Field Summary

Fields inherited from class java.util.EventObject
source

Constructor Summary

Handles_Event(java.lang.Object eventSource) Constructor for Handles_Event

Method Summary

Methods inherited from class java.util.EventObject
getSource, toString

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

Handles_Event

public **Handles_Event**(java.lang.Object eventSource)

Constructor for Handles_Event

Parameters:

eventSource - the object which would fire this event

---

cvm.ucm.handlers.event

Class MediaInitiatorEnableReply_Event

java.lang.Object
  └ java.util.EventObject
    └ cvm.ucm.handlers.event.Handles_Event
      └ **cvm.ucm.handlers.event.MediaInitiatorEnableReply_Event**

All Implemented Interfaces:
java.io.Serializable

```
public class MediaInitiatorEnableReply_Event
extends Handles_Event
```

This class is usd for notifiying a MediaInitiatorEnableReply_Event to the SE.

Author:
Frank Hernandez
See Also:
Serialized Form

Field Summary

private boolean	**notifyEnableMediaReply** the variable holding whether the media is enabled successfully or not

Fields inherited from class java.util.EventObject
source

Constructor Summary

MediaInitiatorEnableReply_Event(java.lang.Object eventSource, boolean notifyEnableMedia)

Method Summary

boolean	**getNotifyMedia**()

Methods inherited from class java.util.EventObject
getSource, toString

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

notifyEnableMediaReply

```
private boolean notifyEnableMediaReply
```

the variable holding whether the media is enabled successfully or not

Constructor Detail

MediaInitiatorEnableReply_Event

```
public MediaInitiatorEnableReply_Event(java.lang.Object eventSource,
                                       boolean notifyEnableMedia)
```

Method Detail

getNotifyMedia

```
public boolean getNotifyMedia()
```

---

cvm.ucm.handlers.event

Class NotifyLoginReply_Event

```
java.lang.Object
  └java.util.EventObject
      └cvm.ucm.handlers.event.Handles_Event
          └cvm.ucm.handlers.event.NotifyLoginReply_Event
```

All Implemented Interfaces:

java.io.Serializable

```
public class NotifyLoginReply_Event
extends Handles_Event
```

This class will represent an event coming from NCB that encapsulate the reply of the login request

Author:

Guangqiang Zhao

See Also:

Serialized Form

Field Summary

private java.lang.Object	**contactList** The contact list of the logged in user

Fields inherited from class java.util.EventObject
source

Constructor Summary

NotifyLoginReply_Event(java.lang.Object eventSource, java.lang.Object reply) The constructor

Method Summary

java.lang.Object	**getContactList**() This method returns the contact list of the user

Methods inherited from class java.util.EventObject

getSource, toString

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

contactList

```
private java.lang.Object contactList
```

The contact list of the logged in user

Constructor Detail

NotifyLoginReply_Event

```
public NotifyLoginReply_Event(java.lang.Object eventSource,
                              java.lang.Object reply)
```

The constructor

Parameters:

`eventSource` - the object that fires this event

`reply` - reply message containing the contact list

Method Detail

getContactList

```
public java.lang.Object getContactList()
```

This method returns the contact list of the user

Returns:

the contact list of the user

---

cvm.ucm.handlers.event

Class NotifyLogoffReply_Event

```
java.lang.Object
  └java.util.EventObject
      └cvm.ucm.handlers.event.Handles_Event
          └cvm.ucm.handlers.event.NotifyLogoffReply_Event
```

All Implemented Interfaces:

java.io.Serializable

```
public class NotifyLogoffReply_Event
extends Handles_Event
```

This event notifies the logoff the system.

Author:
Frank Hernandez

See Also:
Serialized Form

Field Summary

private boolean	**isLogedoff**

Fields inherited from class java.util.EventObject

source

Constructor Summary

NotifyLogoffReply_Event(java.lang.Object eventSource, boolean isLogedoff)

Method Summary

boolean	**isLogedoff**() Return wether or not logoff was successful.

Methods inherited from class java.util.EventObject

getSource, toString

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

isLogedoff

```
private boolean isLogedoff
```

Constructor Detail

NotifyLogoffReply_Event

```
public NotifyLogoffReply_Event(java.lang.Object eventSource,
                               boolean isLogedoff)
```

Method Detail

isLogedoff

```
public boolean isLogedoff()
```

Return wether or not logoff was successful.

Returns:

---

cvm.ucm.handlers.event

Class PartyAddedReply_Event

```
java.lang.Object
  └java.util.EventObject
      └cvm.ucm.handlers.event.Handles_Event
          └cvm.ucm.handlers.event.PartyAddedReply_Event
```

All Implemented Interfaces:

java.io.Serializable

```
public class PartyAddedReply_Event
extends Handles_Event
```

This class will represent an event coming from NCB that encapsulate the reply of the add party request

Author:

Guangqiang Zhao

See Also:

Serialized Form

Field Summary

private boolean	**notifyPartyAdded** The variable holding whether the party is added or not

Fields inherited from class java.util.EventObject
source

Constructor Summary

PartyAddedReply_Event(java.lang.Object eventSource, boolean notifyPartyAdded) The constructor

Method Summary

boolean	**getNotifyPartyAdded**() This method returns a value indicating whether the party is added successfully or not

Methods inherited from class java.util.EventObject
getSource, toString

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

notifyPartyAdded

```
private boolean notifyPartyAdded
```

The variable holding whether the party is added or not

Constructor Detail

PartyAddedReply_Event

```
public PartyAddedReply_Event(java.lang.Object eventSource,
                             boolean notifyPartyAdded)
```

The constructor

Parameters:

`eventSource` - the object that fires this event

`notifyPartyAdded` - containing the result of adding the party

Method Detail

getNotifyPartyAdded

```
public boolean getNotifyPartyAdded()
```

This method returns a value indicating whether the party is added successfully or not

Returns:

a boolean value indicating whether the party is added successfully or not

---

cvm.ucm.handlers.event

Class PartyRemovedReply_Event

```
java.lang.Object
  └java.util.EventObject
      └cvm.ucm.handlers.event.Handles_Event
```

└ **cvm.ucm.handlers.event.PartyRemovedReply_Event**

All Implemented Interfaces:

java.io.Serializable

```
public class PartyRemovedReply_Event
extends Handles_Event
```

This class will represent an event coming from NCB that encapsulate the reply of the remove party request

Author:

Guangqiang Zhao

See Also:

Serialized Form

Field Summary

private boolean	**notifyPartyRemovedReply** The variable holding whether the party is removed or not

Fields inherited from class java.util.EventObject
source

Constructor Summary

PartyRemovedReply_Event(java.lang.Object eventSource, boolean notifyPartyRemovedReply) The constructor

Method Summary

boolean	**getNotifyPartyRemoved**() This method returns a value indicating whether the party is removed successfully or not

Methods inherited from class java.util.EventObject
getSource, toString

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

notifyPartyRemovedReply

```
private boolean notifyPartyRemovedReply
```

The variable holding whether the party is removed or not

Constructor Detail

PartyRemovedReply_Event

```
public PartyRemovedReply_Event(java.lang.Object eventSource,
                               boolean notifyPartyRemovedReply)
```

The constructor

Parameters:

`eventSource` - the object that fires this event

`notifyPartyRemovedReply` - containing the result of removing the party

Method Detail

getNotifyPartyRemoved

```
public boolean getNotifyPartyRemoved()
```

This method returns a value indicating whether the party is removed successfully or not

Returns:

a boolean value indicating whether the party is removed successfully or not

---

cvm.ucm.handlers.event

Class SendSchemaReply_Event

```
java.lang.Object
  └java.util.EventObject
      └cvm.ucm.handlers.event.Handles_Event
          └cvm.ucm.handlers.event.SendSchemaReply_Event
```

All Implemented Interfaces:

java.io.Serializable

```
public class SendSchemaReply_Event
extends Handles_Event
```

This class will represent an event coming from NCB that encapsulate the reply of the sendSchema request

Author:

Guangqiang Zhao

See Also:

Serialized Form

Field Summary

private	**sendStatus**

boolean	A boolean variable holding if the schema is sent successfully

Fields inherited from class java.util.EventObject
source

Constructor Summary

SendSchemaReply_Event(java.lang.Object eventSource, boolean sendStatus) The constructor

Method Summary

boolean	**isSendStatus**() This method returns the status of the schema sent

Methods inherited from class java.util.EventObject
getSource, toString

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

sendStatus

```
private boolean sendStatus
```

A boolean variable holding if the schema is sent successfully

Constructor Detail

SendSchemaReply_Event

```
public SendSchemaReply_Event(java.lang.Object eventSource,
                             boolean sendStatus)
```

The constructor

Parameters:

eventSource - the object that fires this event

sendStatus - indicating if the status of the sent schema

Method Detail

isSendStatus

```
public boolean isSendStatus()
```

This method returns the status of the schema sent

Returns:

the status of the schema sent

---

cvm.ucm.handlers.exception
Class ControlSchemaNotSentException

```
java.lang.Object
  └java.lang.Throwable
      └java.lang.Exception
          └cvm.ucm.handlers.exception.ControlSchemaNotSentException
```

All Implemented Interfaces:

java.io.Serializable

```
public class ControlSchemaNotSentException
extends java.lang.Exception
```

Exception thrown when the control schema passed could not be sent.

Author:

Frank Hernandez

See Also:

Serialized Form

Constructor Summary

`ControlSchemaNotSentException()`

Method Summary

Methods inherited from class java.lang.Throwable
`fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString`

Methods inherited from class java.lang.Object
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

ControlSchemaNotSentException

```
public ControlSchemaNotSentException()
```

---

cvm.ucm.handlers.exception
Class DataNotFoundException

```
java.lang.Object
  └java.lang.Throwable
      └java.lang.Exception
          └cvm.ucm.handlers.exception.DataNotFoundException
```

All Implemented Interfaces:

java.io.Serializable

```
public class DataNotFoundException
extends java.lang.Exception
```

The DataNotFoundException is thrown when the program tries to send a file or a form that is not found

Author:

Guangqiang Zhao

See Also:

Serialized Form

Field Summary

private java.lang.String	**dataName** The name of the data which could not be found

Constructor Summary

DataNotFoundException(java.lang.String name)

Method Summary

java.lang.String	**getDataName**() This method returns the name of the file or form

Methods inherited from class java.lang.Throwable
fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait,

wait, wait

Field Detail

dataName

```
private java.lang.String dataName
```

The name of the data which could not be found

Constructor Detail

DataNotFoundException

```
public DataNotFoundException(java.lang.String name)
```

Method Detail

getDataName

```
public java.lang.String getDataName()
```

This method returns the name of the file or form

Returns:

the name of the file or form

---

cvm.ucm.handlers.exception

Class DataSchemaNotSentException

```
java.lang.Object
  └java.lang.Throwable
      └java.lang.Exception
          └cvm.ucm.handlers.exception.DataSchemaNotSentException
```

All Implemented Interfaces:

java.io.Serializable

```
public class DataSchemaNotSentException
extends java.lang.Exception
```

Exception thrown when the data schema passed could not be sent.

Author:

Frank Hernandez

See Also:

Serialized Form

Constructor Summary

DataSchemaNotSentException()

Method Summary

Methods inherited from class java.lang.Throwable
fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

DataSchemaNotSentException

```
public DataSchemaNotSentException()
```

---

cvm.ucm.handlers.exception

Class IllegalMacroArgumentException

```
java.lang.Object
  └java.lang.Throwable
      └java.lang.Exception
          └cvm.ucm.handlers.exception.IllegalMacroArgumentException
```

All Implemented Interfaces:

java.io.Serializable

```
public class IllegalMacroArgumentException
extends java.lang.Exception
```

This class represents the exception thrown by the macro interpreter when an illegal argument is encountered

Author:

Guangqiang Zhao

See Also:

Serialized Form

Field Summary

private java.lang.String	**argName** the name of the argument that courses this exception
private java.lang.String	**macroName**

	the name of the macro that encounters an unexpected argument

Constructor Summary

`IllegalMacroArgumentException(java.lang.String macro, java.lang.String arg)`

Method Summary

`java.lang.String`	`getIllegalArgument()` This method returns the name of the argument
`java.lang.String`	`getMacroName()` This method returns the name of the macro

Methods inherited from class java.lang.Throwable
`fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString`

Methods inherited from class java.lang.Object
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Field Detail

macroName

`private java.lang.String macroName`

the name of the macro that encounters an unexpected argument

---

argName

`private java.lang.String argName`

the name of the argument that courses this exception

Constructor Detail

IllegalMacroArgumentException

```
public IllegalMacroArgumentException(java.lang.String macro,
                                     java.lang.String arg)
```

Method Detail

getMacroName

`public java.lang.String getMacroName()`

This method returns the name of the macro

Returns:
the name of the macro that encounters an unexpected argument

---

getIllegalArgument

`public java.lang.String getIllegalArgument()`

This method returns the name of the argument

Returns:
the name of the argument that courses this exception

---

cvm.ucm.handlers.exception
Class InvalidScriptException

```
java.lang.Object
  └java.lang.Throwable
      └java.lang.Exception
          └cvm.ucm.handlers.exception.InvalidScriptException
```

All Implemented Interfaces:
java.io.Serializable

```
public class InvalidScriptException
extends java.lang.Exception
```

This class represents the exception thrown by the script interpreter when an invalid script is encountered

Author:
Guangqiang Zhao

See Also:
Serialized Form

Field Summary

private java.lang.String	**scriptName** the name of the script that has invalid syntax

Constructor Summary

InvalidScriptException(java.lang.String name)

Method Summary

java.lang.String	**getscriptName**() This method returns the name of the invalid script

Methods inherited from class java.lang.Throwable
fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

scriptName

```
private java.lang.String scriptName
```

the name of the script that has invalid syntax

Constructor Detail

InvalidScriptException

```
public InvalidScriptException(java.lang.String name)
```

Method Detail

getscriptName

```
public java.lang.String getscriptName()
```

This method returns the name of the invalid script

Returns:

the name of the invalid script

---

cvm.ucm.handlers.exception

Class LoginException

```
java.lang.Object
  └java.lang.Throwable
      └java.lang.Exception
          └cvm.ucm.handlers.exception.LoginException
```

All Implemented Interfaces:

java.io.Serializable

```
public class LoginException
extends java.lang.Exception
```

Exception thrown when the login fails.

Author:

Frank Hernandez

See Also:

Serialized Form

Constructor Summary

LoginException()

Method Summary

Methods inherited from class java.lang.Throwable
fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

LoginException

public **LoginException**()

---

cvm.ucm.handlers.exception

Class MacroNotFoundException

java.lang.Object
  └ java.lang.Throwable
      └ java.lang.Exception
          └ **cvm.ucm.handlers.exception.MacroNotFoundException**

All Implemented Interfaces:

java.io.Serializable

---

public class **MacroNotFoundException**
extends java.lang.Exception

This class represents the exception thrown by the macro loader when a given macro could not be found in the repository

Author:

Guangqiang Zhao

See Also:

Serialized Form

Field Summary

private java.lang.String	**macroName** the name of the missed macro

Constructor Summary

MacroNotFoundException(java.lang.String macro)

Method Summary

java.lang.String	**getMacroName**() This method returns the name of the missed macro

Methods inherited from class java.lang.Throwable
fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

macroName

private java.lang.String **macroName**

the name of the missed macro

Constructor Detail

MacroNotFoundException

public **MacroNotFoundException**(java.lang.String macro)

Method Detail

getMacroName

public java.lang.String **getMacroName**()

This method returns the name of the missed macro

Returns:

name of the missed macro

---

cvm.ucm.handlers.exception
Class NoSessionException

```
java.lang.Object
  └java.lang.Throwable
      └java.lang.Exception
          └cvm.ucm.handlers.exception.NoSessionException
```

All Implemented Interfaces:

java.io.Serializable

```
public class NoSessionException
extends java.lang.Exception
```

The NoSessionException is thrown when the program tries to add parties or send data in a connection that has no corresponding physical sessions

Author:

Guangqiang Zhao

See Also:

Serialized Form

Field Summary

private java.lang.String	**connID** the connection ID which has not physical session in the NCB layer

Constructor Summary

NoSessionException(java.lang.String ID)

Method Summary

java.lang.String	**getConnectionID**() This method returns the connection ID

Methods inherited from class java.lang.Throwable
fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait,

wait, wait

Field Detail

connID

private java.lang.String **connID**

the connection ID which has not physical session in the NCB layer

Constructor Detail

NoSessionException

public **NoSessionException**(java.lang.String ID)

Method Detail

getConnectionID

public java.lang.String **getConnectionID**()

This method returns the connection ID

Returns:

the connection ID that missed a physical session in the low layer

---

cvm.ucm.handlers.exception

Class PartyNotAddedException

java.lang.Object
  └java.lang.Throwable
      └java.lang.Exception
          └**cvm.ucm.handlers.exception.PartyNotAddedException**

All Implemented Interfaces:

java.io.Serializable

---

public class **PartyNotAddedException**
extends java.lang.Exception

This exception occurs when the adding of a party fails.

Author:

Frank Hernandez

See Also:

Serialized Form

Field Summary

private java.lang.String	**partyName** the name of the participant that coursed this exception

Constructor Summary

`PartyNotAddedException(java.lang.String name)`

Method Summary

`java.lang.String`	`getPartyName()` This method returns the name of the participant

Methods inherited from class java.lang.Throwable

`fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString`

Methods inherited from class java.lang.Object

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Field Detail

partyName

`private java.lang.String partyName`

the name of the participant that coursed this exception

Constructor Detail

PartyNotAddedException

`public PartyNotAddedException(java.lang.String name)`

Method Detail

getPartyName

`public java.lang.String getPartyName()`

This method returns the name of the participant

Returns:

the name of the participant

---

cvm.ucm.handlers.exception

Class PartyNotFoundException

```
java.lang.Object
  └java.lang.Throwable
      └java.lang.Exception
          └cvm.ucm.handlers.exception.PartyNotFoundException
```

All Implemented Interfaces:

java.io.Serializable

```
public class PartyNotFoundException
extends java.lang.Exception
```

This class represents an exception that is thrown when the program tries to remove a party that is not in the connection

Author:

Guangqiang Zhao

See Also:

Serialized Form

Field Summary

private java.lang.String	**partyName** the name of the participant that coursed this exception

Constructor Summary

PartyNotFoundException(java.lang.String name)

Method Summary

java.lang.String	**getPartyName**() This method returns the name of the participant

Methods inherited from class java.lang.Throwable
fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

partyName

```
private java.lang.String partyName
```

the name of the participant that coursed this exception

Constructor Detail

PartyNotFoundException

```
public PartyNotFoundException(java.lang.String name)
```

Method Detail

getPartyName

```
public java.lang.String getPartyName()
```

This method returns the name of the participant

Returns:

the name of the participant

---

cvm.ucm.handlers.exception

Class SchemaNotSavedException

```
java.lang.Object
  └java.lang.Throwable
      └java.lang.Exception
          └cvm.ucm.handlers.exception.SchemaNotSavedException
```

All Implemented Interfaces:

java.io.Serializable

```
public class SchemaNotSavedException
extends java.lang.Exception
```

Exception thrown when the schema could not be saved.

Author:

Frank Hernandez

See Also:

Serialized Form

Constructor Summary

SchemaNotSavedException()

Method Summary

Methods inherited from class java.lang.Throwable
fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

SchemaNotSavedException

public **SchemaNotSavedException**()

---

cvm.ucm.interpreter
Interface Command

All Known Implementing Classes:

MacroCommand

---

public interface **Command**

Command Interface This is the command interface as in the Command Design Patern from GoF.

Author:

Frank Hernandez

Method Summary

void	**execute**()

Method Detail

execute

void **execute**()
throws java.lang.Exception

Throws:

java.lang.Exception

---

cvm.ucm.interpreter
Class EnvVariable

java.lang.Object
  └ **cvm.ucm.interpreter.EnvVariable**

---

public class **EnvVariable**
extends java.lang.Object

This class is used to hold the 'enviroment variables' needed by the macro to execute. For a macro to use a system variable in the system it must know the variable name as in the macro, the variable type (full declaration i.e. Java.lang.util.String) and the variable value (the variable name as seen in the class).

Author:
Frank Hernandez

Field Summary

private java.lang.String	**typeName**
private java.lang.String	**varName**
private java.lang.Object	**varValue**

Constructor Summary

EnvVariable(java.lang.String varName, java.lang.String typeName, java.lang.Object varValue) Creates a new enviroment variable object.

Method Summary

java.lang.String	**getParamName**() This method returns the name of the variable as seen inside the macro.
java.lang.String	**getParamType**() This method returns the full classification of the variable.
java.lang.Object	**getParamValue**() This method returns the value to be given to the variable inside the macro.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

varName

private java.lang.String **varName**

---

typeName
```
private java.lang.String typeName
```

---

varValue
```
private java.lang.Object varValue
```

Constructor Detail

EnvVariable
```
public EnvVariable(java.lang.String varName,
                   java.lang.String typeName,
                   java.lang.Object varValue)
```
Creates a new enviroment variable object.

Parameters:

varName -

typeName -

varValue -

Method Detail

getParamName
```
public java.lang.String getParamName()
```
This method returns the name of the variable as seen inside the macro.

Returns:

variable name as seen in the macro.

---

getParamType
```
public java.lang.String getParamType()
```
This method returns the full classification of the variable.

Returns:

type name of the variable i.e. 'cvm.ncb.NetworkCommunicationBroker'

---

getParamValue
```
public java.lang.Object getParamValue()
```
This method returns the value to be given to the variable inside the macro. That is the variable as used inside the system.

Returns:

the value to be given to the variable inside the macro.

---

cvm.ucm.interpreter
Class MacroCommand
```
java.lang.Object
  └ cvm.ucm.interpreter.MacroCommand
```
All Implemented Interfaces:

Command

```
public class MacroCommand
extends java.lang.Object
implements Command
```

Command Encapsulation for Macro execution. The MacroCommand will incapsulate a Macro for later execution. The Macro comes in the form of a MacroNode that contains a ScriptInterpreter and the list of values to execute.

Author:
Frank Hernandez

See Also:
MacroNode

Field Summary

private java.lang.String	**macroName**
private MacroNode	**myMacro**
private UCMExceptionHandler	**ucmException**

Constructor Summary

MacroCommand(MacroNode node, java.lang.String sName)

Method Summary

void	**execute**() This is the implementation of the Command execute method.
java.lang.String	**getName**() Retruens the Name of the Macro to Execute.
MacroNode	**getNode**() Returns the Macro Node.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

myMacro
```
private MacroNode myMacro
```

---

macroName
```
private java.lang.String macroName
```

---

ucmException
```
private UCMExceptionHandler ucmException
```

Constructor Detail

MacroCommand
```
public MacroCommand(MacroNode node,
                    java.lang.String sName)
```

Method Detail

execute
```
public void execute()
            throws java.lang.Exception
```
This is the implementation of the Command execute method. This method executes the Command by evalueting the ScriptInterpreter inside the MacroNode with the parameter values.

Specified by:

execute in interface Command

Throws:

java.lang.Exception

See Also:

MacroNode

---

getNode
```
public MacroNode getNode()
```
Returns the Macro Node.

---

getName
```
public java.lang.String getName()
```
Retruens the Name of the Macro to Execute.

Returns:

---

cvm.ucm.interpreter
Class MacroInterpreter
```
java.lang.Object
  └cvm.ucm.interpreter.MacroInterpreter
```

```
public class MacroInterpreter
extends java.lang.Object
```

MacroInterpreter encapsulates the creation of a script evaluator object which will be executed using Janino inside the MacroCommand execute. The main purpose of this class is to receive a collection of parameter values and a macro object, which will contain parsed macro information. The MacroInterpreter will then encapsulate the data in a MacroNode for later execution.

Author:
Raidel Batista

See Also:
ScriptEvaluator, MacroNode, MacroCommand

Constructor Summary

MacroInterpreter()

Method Summary

protected MacroNode	**interpretMacro**(Macro repoMacro, java.util.ArrayList paramValuesArr, java.util.ArrayList<EnvVariable> envVars) The main purpose of this method is to receive a collection of parameter values and a macro object, which will contain parsed macro information.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

MacroInterpreter

```
public MacroInterpreter()
```

Method Detail

interpretMacro

```
protected MacroNode interpretMacro(Macro repoMacro,
                                   java.util.ArrayList paramValuesArr,
                                   java.util.ArrayList<EnvVariable> envVars)
                            throws java.lang.NoSuchMethodException,
                                   java.lang.InstantiationException,
                                   java.lang.reflect.InvocationTargetException,
                                   java.lang.IllegalAccessException,
                                   org.codehaus.janino.CompileException,
```

`org.codehaus.janino.Parser.ParseException`,

`org.codehaus.janino.Scanner.ScanException`

The main purpose of this method is to receive a collection of parameter values and a macro object, which will contain parsed macro information. The MacroInterpreter will then encapsulate the data in a MacroNode for later execution.

Parameters:

`repoMacro` -

`paramValues` -

`sEngine` -

`ncb` -

Returns:

A MacroNode to be used in the creation of a MacroCommand

Throws:

`java.lang.NoSuchMethodException`
`java.lang.InstantiationException`
`java.lang.reflect.InvocationTargetException`
`java.lang.IllegalAccessException`
`org.codehaus.janino.CompileException`
`org.codehaus.janino.Parser.ParseException`
`org.codehaus.janino.Scanner.ScanException`

---

cvm.ucm.interpreter
Class MacroNode

`java.lang.Object`
  └ **`cvm.ucm.interpreter.MacroNode`**

`public class` **`MacroNode`**
`extends java.lang.Object`

MacroNode class This class encapsulates all the data needed for the execution of the ScriptEvaluator object. There is one MacroNode per macro to be executed.

Author:

Frank Hernandez

See Also:

`ScriptEvaluator`

Field Summary

private `org.codehaus.janino.ScriptEvaluator`	**`mySE`**
private `java.lang.Object[]`	**`paramVals`**

Constructor Summary

`MacroNode(org.codehaus.janino.ScriptEvaluator se, java.lang.Object[] params)`

Method Summary

`java.lang.Object[]`	`getParameterValues()` This accessor method returns Returns the array of parameter values.
`org.codehaus.janino.ScriptEvaluator`	`getSE()` This accessor method returns a ScrpitEvaluator object.

Methods inherited from class java.lang.Object

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Field Detail

mySE

```
private org.codehaus.janino.ScriptEvaluator mySE
```

---

paramVals

```
private java.lang.Object[] paramVals
```

Constructor Detail

MacroNode

```
public MacroNode(org.codehaus.janino.ScriptEvaluator se,
                 java.lang.Object[] params)
```

Method Detail

getSE

```
public org.codehaus.janino.ScriptEvaluator getSE()
```

This accessor method returns a ScrpitEvaluator object.

Returns:

ScrpitEvaluator object

See Also:

`ScriptEvaluator`

---

getParameterValues

```
public java.lang.Object[] getParameterValues()
```

This accessor method returns Returns the array of parameter values.

Returns:
array of parameter value;

---

cvm.ucm.interpreter
Class Parser

```
java.lang.Object
  └ cvm.ucm.interpreter.Parser
```

```
public class Parser
extends java.lang.Object
```

This class parses the control scripts that are passed down from the Synthesis Engine and puts them in a data structure that is easier to handle and understand.

Author:
Eduardo Monteiro

Field Summary

private java.io.BufferedReader	reader

Constructor Summary

Parser()	

Method Summary

private java.lang.String	getFunctionName(java.lang.String line) Gets the function name of a given line of the control script.
private java.util.ArrayList	getParmTypes(java.lang.String line) Gets a list of parameter types from a function in a given line of the control script
private java.util.ArrayList	getParmValues(java.lang.String line) Gets a list of parameter values from a function in a given line of the control script
private java.lang.String	getReturnType(java.lang.String line) Gets the return type of a function in a given line of the control script
static void	main(java.lang.String[] args)

static java.util.ArrayList	**makeList**(java.lang.String stringList)
Script	**parse**(java.lang.String script)

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

reader

private java.io.BufferedReader **reader**

Constructor Detail

Parser

public **Parser**()

Method Detail

makeList

public static java.util.ArrayList **makeList**(java.lang.String stringList)

---

parse

public Script **parse**(java.lang.String script)
throws InvalidScriptException

Parameters:

String - representing the control script

Returns:

Script object containing the parsed control script

Throws:

InvalidScriptException

---

getFunctionName

private java.lang.String **getFunctionName**(java.lang.String line)

Gets the function name of a given line of the control script.

Parameters:

String - representing one line of the parsed control script

Returns:

String representing the function name.

---

getReturnType

private java.lang.String **getReturnType**(java.lang.String line)

Gets the return type of a function in a given line of the control script

Parameters:
`String` - representing one line of the parsed control script
Returns:
String representing the return type.

---

getParmTypes

`private java.util.ArrayList` **`getParmTypes`**`(java.lang.String line)`

Gets a list of parameter types from a function in a given line of the control script
Parameters:
`String` - representing one line of the parsed control script
Returns:
ArrayList with the parameter types.

---

getParmValues

`private java.util.ArrayList` **`getParmValues`**`(java.lang.String line)`

Gets a list of parameter values from a function in a given line of the control script
Parameters:
`String` - representing one line of the parsed control script
Returns:
ArrayList with the parameter values.

---

cvm.ucm.interpreter
Class Script

```
java.lang.Object
  └ cvm.ucm.interpreter.Script
```

```
public class Script
extends java.lang.Object
```

Data structure that holds a control script in memory.

Author:
Eduardo Monteiro

Nested Class Summary

class	**Script.Call** Call holds function calls in a given control script.

Field Summary

private java.util.ArrayList	**callList**
private java.util.Iterator	**itr**

Constructor Summary

Script() Constructor	

Method Summary

boolean	**add**(java.lang.String functionName, java.lang.String returnType, java.util.ArrayList parmTypes, java.util.ArrayList parmValues) Adds a new call to the script data structure.
Script.Call	**getNextCall**() Retrieves the next available call in the control script.
boolean	**hasNext**() Checks if the script has any more function calls.
java.lang.String	**toString**()

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

callList

private java.util.ArrayList **callList**

---

itr

private java.util.Iterator **itr**

Constructor Detail

Script

public **Script**()

Constructor

Method Detail

hasNext

public boolean **hasNext**()

Checks if the script has any more function calls.

Returns:

True if script has more calls, False otherwise.

---

getNextCall

```
public Script.Call getNextCall()
```

Retrieves the next available call in the control script.

Returns:

Call object representing the next available function call.

---

add

```
public boolean add(java.lang.String functionName,
                   java.lang.String returnType,
                   java.util.ArrayList parmTypes,
                   java.util.ArrayList parmValues)
```

Adds a new call to the script data structure.

Parameters:

`String` - functionName, name of the function

`String` - returnType, return type of the function

`ArrayList` - parmTypes, function parameter types

`ArrayList` - parmValues, function parameter values

Returns:

True if added successfuly, False otherwise.

---

toString

```
public java.lang.String toString()
```

Overrides:

`toString` in class `java.lang.Object`

---

cvm.ucm.interpreter
Class UCM_I_Facade

```
java.lang.Object
  └ cvm.ucm.interpreter.UCM_I_Facade
```

```
public class UCM_I_Facade
extends java.lang.Object
```

Façade class that provides an interface into the UCM_Interpreter subsystem. The purpose of this class is to aid future expansion of this subsystem. OCL Statement: Context: UCM_I_Facade inv: ucmAdapter<>null

Author:

Frank Hernandez

Field Summary

private UCM_Interpreter_Adapter	**ucmAdapter**

Constructor Summary

UCM_I_Facade()

Method Summary

void	**attachNCB**(NetworkCommunicationBroker thisNcb) Attaches the NCB instance to the CM_Interepreter_Mk.
void	**attachSynthesisEngine**(SynthesisEngine thisSE) Attaches the SynthesisEngine intance to the CM_Interepreter_Mk.
void	**executeNext**() This method singals the Microkernel to continue and execute the next command.
void	**parseScript**(java.lang.String script) This method executes a control scipt received from the SynthesisEngine.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

ucmAdapter

private UCM_Interpreter_Adapter **ucmAdapter**

Constructor Detail

UCM_I_Facade

public **UCM_I_Facade**()

Method Detail

parseScript

public void **parseScript**(java.lang.String script)

This method executes a control scipt received from the SynthesisEngine. OCL Statement: Context: UCM_I_Facade::parseScript(script, se,ncb) pre: script.length>0 and se<>null and ncb<>nul Context: UCM_I_Facade::parseScript(script, se,ncb) post: ucmAdapter<>null

Parameters:

script -

se -

ncb -

Throws:

org.codehaus.janino.CompileException

org.codehaus.janino.Parser.ParseException
org.codehaus.janino.Scanner.ScanException
java.lang.NoSuchMethodException
java.lang.InstantiationException
java.lang.reflect.InvocationTargetException
java.lang.IllegalAccessException

---

attachNCB

`public void attachNCB(NetworkCommunicationBroker thisNcb)`

Attaches the NCB instance to the CM_Interepreter_Mk.

Parameters:

`thisNcb` -

---

attachSynthesisEngine

`public void attachSynthesisEngine(SynthesisEngine thisSE)`

Attaches the SynthesisEngine intance to the CM_Interepreter_Mk.

Parameters:

`thisSE` -

---

executeNext

`public void executeNext()`

This method singals the Microkernel to continue and execute the next command.

---

cvm.ucm.interpreter

Class UCM_Interepreter_Mk

```
java.lang.Object
  └ cvm.ucm.interpreter.UCM_Interepreter_Mk
```

```
public class UCM_Interepreter_Mk
extends java.lang.Object
```

This is the microkernel as in the architecture pattern. This class control the parsing of scripts, the loading of macros, and the queuing and execution of commands.

Author:

Frank Hernandez

Field Summary

private java.util.Queue<MacroCommand>	**commandQueue**
private java.util.ArrayList<EnvVariable>	**envVars**
private static UCM_Interepreter_Mk	**instance**

private java.util.Iterator	**itr**
private NetworkCommunicationBroker	**ncb**
private NCBEventObjectManager	**ncbNotifier**
private Parser	**scriptParser**
private SynthesisEngine	**se**
private java.util.Map<java.lang.String,java.lang.String>	**ucmConnMap**
private UCMEventHandler	**ucmEHandler**
private UCMEventObjectManager	**ucmNotifier**
private UCMExceptionHandler	**ucmXHandler**

Constructor Summary

private	**UCM_Interepreter_Mk**()

Method Summary

void	**attachNCB**(NetworkCommunicationBroker thisNcb) Attaches the NCB instance to the CM_Interepreter_Mk.
void	**attachSynthesisEngine**(SynthesisEngine thisSE) Attaches the SynthesisEngine intance to the CM_Interepreter_Mk.
void	**executeNextCommand**() This method excecutes the next command in line.
static UCM_Interepreter_Mk	**Instance**() This is the Instance implementation of Singleton Design Pattern.
void	**parseScript**(java.lang.String script) This method parses and executes a control scipt received from the SynthesisEngine.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

instance
private static UCM_Interepreter_Mk **instance**

---

scriptParser
private Parser **scriptParser**

---

commandQueue
private java.util.Queue<MacroCommand> **commandQueue**

---

envVars
private java.util.ArrayList<EnvVariable> **envVars**

---

ncb
private NetworkCommunicationBroker **ncb**

---

se
private SynthesisEngine **se**

---

ucmNotifier
private UCMEventObjectManager **ucmNotifier**

---

ncbNotifier
private NCBEventObjectManager **ncbNotifier**

---

ucmXHandler
private UCMExceptionHandler **ucmXHandler**

---

ucmEHandler
private UCMEventHandler **ucmEHandler**

---

ucmConnMap
private java.util.Map<java.lang.String,java.lang.String> **ucmConnMap**

---

itr
private java.util.Iterator **itr**

Constructor Detail

UCM_Interepreter_Mk

`private UCM_Interepreter_Mk()`

Method Detail

parseScript

`public void parseScript(java.lang.String script)`

This method parses and executes a control scipt received from the SynthesisEngine.

Parameters:

`script` -

`se` -

`ncb` -

Throws:

`org.codehaus.janino.CompileException`
`org.codehaus.janino.Parser.ParseException`
`org.codehaus.janino.Scanner.ScanException`
`java.lang.NoSuchMethodException`
`java.lang.InstantiationException`
`java.lang.reflect.InvocationTargetException`
`java.lang.IllegalAccessException`

Instance

`public static UCM_Interepreter_Mk Instance()`

This is the Instance implementation of Singleton Design Pattern.

Returns:

An instance of UCM_Interepreter_Mk

---

attachNCB

`public void attachNCB(NetworkCommunicationBroker thisNcb)`

Attaches the NCB instance to the CM_Interepreter_Mk.

Parameters:

`thisNcb` -

---

attachSynthesisEngine

`public void attachSynthesisEngine(SynthesisEngine thisSE)`

Attaches the SynthesisEngine intance to the CM_Interepreter_Mk.

Parameters:

`thisSE` -

---

executeNextCommand

`public void executeNextCommand()`

This method excecutes the next command in line. Used for steping command Macro execution.

See Also:

`MacroCommand`

---

cvm.ucm.interpreter

Class UCM_Interpreter_Adapter

```
java.lang.Object
  └ cvm.ucm.interpreter.UCM_Interpreter_Adapter
```

```
public class UCM_Interpreter_Adapter
extends java.lang.Object
```

Adapter class as in the microkernel architecture pattern. It makes the parse script requests to the UCM_Interepreter_Mk class.

Author:

Frank Hernandez

Field Summary

private static UCM_Interpreter_Adapter	**instance**
private static UCM_Interepreter_Mk	**ucmMk**

Constructor Summary

private	**UCM_Interpreter_Adapter**()

Method Summary

void	**attachNCB**(NetworkCommunicationBroker thisNcb) Attaches the NCB instance to the CM_Interepreter_Mk.
void	**attachSynthesisEngine**(SynthesisEngine thisSE) Attaches the SynthesisEngine intance to the CM_Interepreter_Mk.
void	**executeNext**() This method singals the Microkernel to continue and execute the next command.
static UCM_Interpreter_Adapter	**Instance**() This is the Instance implementation of Singleton Design Pattern.
void	**parseScript**(java.lang.String script) This method parses and executes a control scipt received from the SynthesisEngine.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

instance

```
private static UCM_Interpreter_Adapter instance
```

---

ucmMk

```
private static UCM_Interepreter_Mk ucmMk
```

Constructor Detail

UCM_Interpreter_Adapter

```
private UCM_Interpreter_Adapter()
```

Method Detail

parseScript

```
public void parseScript(java.lang.String script)
```

This method parses and executes a control scipt received from the SynthesisEngine.

Parameters:

`script` -

`se` -

`ncb` -

Throws:

`org.codehaus.janino.CompileException`
`org.codehaus.janino.Parser.ParseException`
`org.codehaus.janino.Scanner.ScanException`
`java.lang.NoSuchMethodException`
`java.lang.InstantiationException`
`java.lang.reflect.InvocationTargetException`
`java.lang.IllegalAccessException`

Instance

```
public static UCM_Interpreter_Adapter Instance()
```

This is the Instance implementation of Singleton Design Pattern.

Returns:

An instance of UCM_Interpreter_Adapter

---

attachNCB

```
public void attachNCB(NetworkCommunicationBroker thisNcb)
```

Attaches the NCB instance to the CM_Interepreter_Mk.

Parameters:

`thisNcb` -

attachSynthesisEngine

`public void attachSynthesisEngine(SynthesisEngine thisSE)`

Attaches the SynthesisEngine intance to the CM_Interepreter_Mk.

Parameters:

`thisSE` -

executeNext

`public void executeNext()`

This method singals the Microkernel to continue and execute the next command.

cvm.ucm.manager

Class UCM_M_Facade

```
java.lang.Object
  └ cvm.ucm.manager.UCM_M_Facade
```

```
public class UCM_M_Facade
extends java.lang.Object
```

Façade class that provides an interface into the UCM_Manager subsystem. The purpose of this class is to aid future expansion of this subsystem. OCL Statement: Context: UCM_M_Facade inv: ucmManager<>null

Author:

Frank Hernandez

Field Summary

private static UCM_M_Facade	**instance**
private UCMManager	**manager**

Constructor Summary

private	**UCM_M_Facade**(SynthesisEngine se, NetworkCommunicationBroker ncb)

Method Summary

void	**executeScript**(java.lang.String script) This method executes a control scipt received from the

	SynthesisEngine.
static UCM_M_Facade	**Instance**(SynthesisEngine se, NetworkCommunicationBroker ncb)
static void	**main**(java.lang.String[] args) Testing Manager Facade Class - Unit Test
static void	**notifyEvent**(Handles_Event event) Notifies the occurance of an even.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

manager

private UCMManager **manager**

---

instance

private static UCM_M_Facade **instance**

Constructor Detail

UCM_M_Facade

private **UCM_M_Facade**(SynthesisEngine se, NetworkCommunicationBroker ncb)

Method Detail

Instance

public static UCM_M_Facade **Instance**(SynthesisEngine se, NetworkCommunicationBroker ncb)

---

notifyEvent

public static void **notifyEvent**(Handles_Event event)

Notifies the occurance of an even. OCL Statement: Context: UCM_M_Facade::notifyEvent(event) pre: event <> null Context: UCM_M_Facade::notifyEvent(event) post: ucmManager<>null

Parameters:

event - event to handle

---

executeScript

public void **executeScript**(java.lang.String script)

This method executes a control scipt received from the SynthesisEngine. OCL Statement: Context: UCM_M_Facade::executeScript(script) pre: script.length>0 Context: UCM_M_Facade::executeScript(script) post: ucmManager<>null

Parameters:
`script` - to parse.

---

main

```
public static void main(java.lang.String[] args)
```

Testing Manager Facade Class - Unit Test

Parameters:
`args` -

---

cvm.ucm.manager

Class UCMManager

```
java.lang.Object
  └ cvm.ucm.manager.UCMManager
```

```
public class UCMManager
extends java.lang.Object
```

Controls the work flow inside the UCM system. This class also notifies the overlying system of any event that are specific to it.

Author:
Frank Hernandez

Field Summary

private static UCMManager	**instance**
private static UCM_I_Facade	**interpreterF**
private static NetworkCommunicationBroker	**ncb**
private static SynthesisEngine	**se**

Constructor Summary

private	**UCMManager**()

Method Summary

void	**attachNCB**(NetworkCommunicationBroker thisNcb)

	Attaches the NCB instance to the UCMManager.
void	**attachSynthesisEngine**(SynthesisEngine thisSE) Attaches the SynthesisEngine intance to the UCMManager.
void	**executeScript**(java.lang.String script) This method executes a control scipt received from the SynthesisEngine.
static UCMManager	**Instance**() This is the Instance implementation of Singleton Design Pattern.
static void	**notifyEvent**(Handles_Event event) Notifies the occurance of an even.
static void	**notifySEEvent**(Handles_Event e) Notifies the SE of an event.
private static void	**stepExecution**()

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

instance
private static UCMManager **instance**

---

ncb
private static NetworkCommunicationBroker **ncb**

---

se
private static SynthesisEngine **se**

---

interpreterF
private static UCM_I_Facade **interpreterF**

Constructor Detail

UCMManager
private **UCMManager**()

Method Detail

Instance
public static UCMManager **Instance**()

This is the Instance implementation of Singleton Design Pattern.

Returns:

UCMManager instance

---

notifyEvent

```
public static void notifyEvent(Handles_Event event)
```

Notifies the occurance of an even.

Parameters:

`event` -

---

notifySEEvent

```
public static void notifySEEvent(Handles_Event e)
```

Notifies the SE of an event.

Parameters:

`e` -

---

stepExecution

```
private static void stepExecution()
```

---

executeScript

```
public void executeScript(java.lang.String script)
```

This method executes a control scipt received from the SynthesisEngine.

Parameters:

`script` -

Throws:

`org.codehaus.janino.CompileException`
`org.codehaus.janino.Parser.ParseException`
`org.codehaus.janino.Scanner.ScanException`
`java.lang.NoSuchMethodException`
`java.lang.InstantiationException`
`java.lang.reflect.InvocationTargetException`
`java.lang.IllegalAccessException`

---

attachNCB

```
public void attachNCB(NetworkCommunicationBroker thisNcb)
```

Attaches the NCB instance to the UCMManager.

Parameters:

`thisNcb` -

---

attachSynthesisEngine

```
public void attachSynthesisEngine(SynthesisEngine thisSE)
```

Attaches the SynthesisEngine intance to the UCMManager.

Parameters:

`thisSE` -

---

cvm.ucm.model
Class Macro

```
java.lang.Object
  └cvm.ucm.model.Macro
```

```
public class Macro
extends java.lang.Object
```

This class encapsulates all the data specific to any macro. This object simplifies the transfer of information across subsystems dealing with macros.

Author:
Frank Hernandez

Field Summary

private java.lang.String	**name**
private java.util.ArrayList	**paramNameList**
private java.util.ArrayList	**paramTypeList**
private java.lang.String	**returnType**
private java.lang.String	**script**
private java.util.ArrayList	**thrownExceptions**

Constructor Summary

Macro()	

Method Summary

java.lang.String	**getName**() This method returns the name of the macro
java.util.ArrayList	**getParamNameList**() This method returns a list containing the name of the parameters inside a macro.
java.util.ArrayList	**getParamTypeList**() This method returns a list containing the types of all the parameters in a macro.

java.lang.String	**getReturnType**() This method returns the return type of a macro.
java.lang.String	**getScript**() This function returns a string representation of a script.
java.util.ArrayList	**getThrownExceptions**() Returns the list of thrown exceptions.
void	**setName**(java.lang.String name) This sets the name of the macro.
void	**setParamNameList**(java.util.ArrayList paramNameList) This method sets the list of parameter names.
void	**setParamTypeList**(java.util.ArrayList paramTypeList) This method set the list of parameters types of a macro.
void	**setReturnType**(java.lang.String returnType) This method sets the return type of a macro.
void	**setScript**(java.lang.String script) Sets the execution section of a macro.
void	**setThrownExceptions**(java.util.ArrayList thrownExceptions) Assignes the list of thrown excpetions.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

paramNameList
private java.util.ArrayList **paramNameList**

---

paramTypeList
private java.util.ArrayList **paramTypeList**

---

thrownExceptions
private java.util.ArrayList **thrownExceptions**

---

script
private java.lang.String **script**

---

returnType
private java.lang.String **returnType**

---

name
private java.lang.String **name**

Constructor Detail

Macro

```
public Macro()
```

Method Detail

getParamNameList

```
public java.util.ArrayList getParamNameList()
```

This method returns a list containing the name of the parameters inside a macro.

Returns:

List of Macro Parameter Names

---

setParamNameList

```
public void setParamNameList(java.util.ArrayList paramNameList)
```

This method sets the list of parameter names.

Parameters:

`paramNameList` -

---

getParamTypeList

```
public java.util.ArrayList getParamTypeList()
```

This method returns a list containing the types of all the parameters in a macro.

Returns:

List of type of parameters.

---

setParamTypeList

```
public void setParamTypeList(java.util.ArrayList paramTypeList)
```

This method set the list of parameters types of a macro.

Parameters:

`paramTypeList` -

---

getScript

```
public java.lang.String getScript()
```

This function returns a string representation of a script. That is it retuns the execution section of a macro in string format.

Returns:

String contining the execution section of a macro.

---

setScript

```
public void setScript(java.lang.String script)
```

Sets the execution section of a macro.

Parameters:

`script` -

---

getReturnType

```
public java.lang.String getReturnType()
```

This method returns the return type of a macro.

Returns:

The return type of the macro.

---

setReturnType

```
public void setReturnType(java.lang.String returnType)
```

This method sets the return type of a macro.

Parameters:

`returnType` -

---

getName

```
public java.lang.String getName()
```

This method returns the name of the macro

Returns:

Name of the macro.

---

setName

```
public void setName(java.lang.String name)
```

This sets the name of the macro.

Parameters:

`name` -

---

setThrownExceptions

```
public void setThrownExceptions(java.util.ArrayList thrownExceptions)
```

Assignes the list of thrown excpetions.

Parameters:

`thrownExceptions` -

---

getThrownExceptions

```
public java.util.ArrayList getThrownExceptions()
```

Returns the list of thrown exceptions.

Returns:

---

cvm.ucm.repository

Interface Sources

All Known Implementing Classes:

SourceDBLoader, SourceFileSysLoader

```
public interface Sources
```

Interface to Source loaders

Method Summary

Macro	**loader**(java.lang.String name)

Method Detail

loader

Macro **loader**(java.lang.String name)
throws java.lang.Exception

Throws:
java.lang.Exception

---

cvm.ucm.repository

Class MacroLoader

java.lang.Object
└ **cvm.ucm.repository.MacroLoader**

---

public class **MacroLoader**
extends java.lang.Object

The MacroLoader class retrieves, from a given Source (a Database, a File System, etc), the information necessary and creates a Macro.

Author:
Abhishek B. and Marylurdys H.

Field Summary

(package private) Sources	**s**

Constructor Summary

MacroLoader(Sources s)

Method Summary

Macro	**loadMacro**(java.lang.String name) loadMacro creates a Macro object with the information obtained from the source for a given function name

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

s
Sources **s**

Constructor Detail

MacroLoader
public **MacroLoader**(Sources s)

Method Detail

loadMacro
public Macro **loadMacro**(java.lang.String name)
throws java.lang.Exception

loadMacro creates a Macro object with the information obtained from the source for a given function name

Parameters:
name: - the name of the function whose macro is required

Returns:
Macro object

Throws:
java.lang.Exception

See Also:
Macro

---

cvm.ucm.repository
Class MSAccessConnection
java.lang.Object
  └ **cvm.ucm.repository.MSAccessConnection**

---

public class **MSAccessConnection**
extends java.lang.Object

Field Summary

private java.sql.Connection	**conn**
private static java.lang.String	**driver**
private	**password**

java.lang.String	
private java.lang.String	**url**
private java.lang.String	**username**

Constructor Summary

MSAccessConnection(Rep_Properties prop)

Method Summary

java.sql.Connection	**getConn**()

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

username
private java.lang.String **username**

---

password
private java.lang.String **password**

---

url
private java.lang.String **url**

---

driver
private static java.lang.String **driver**

---

conn
private java.sql.Connection **conn**

Constructor Detail

MSAccessConnection
public **MSAccessConnection**(Rep_Properties prop)

Method Detail

getConn

```
public java.sql.Connection getConn()
```

---

cvm.ucm.repository

Class MySQLConnection

```
java.lang.Object
  └ cvm.ucm.repository.MySQLConnection
```

```
public class MySQLConnection
extends java.lang.Object
```

Field Summary

private java.sql.Connection	**conn**
private java.lang.String	**dbName**
private static java.lang.String	**driver**
private java.lang.String	**hostname**
private java.lang.String	**password**
private java.lang.String	**port**
private java.lang.String	**url**
private java.lang.String	**username**

Constructor Summary

MySQLConnection(Rep_Properties prop)

Method Summary

java.sql.Connection	**getConn**()

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

username
private java.lang.String **username**

---

password
private java.lang.String **password**

---

url
private java.lang.String **url**

---

hostname
private java.lang.String **hostname**

---

port
private java.lang.String **port**

---

dbName
private java.lang.String **dbName**

---

driver
private static java.lang.String **driver**

---

conn
private java.sql.Connection **conn**

Constructor Detail

MySQLConnection
public **MySQLConnection**(Rep_Properties prop)

Method Detail

getConn
public java.sql.Connection **getConn**()

---

cvm.ucm.repository

Class Rep_Default_Properties
java.lang.Object
  └java.util.Dictionary<K,V>
    └java.util.Hashtable<java.lang.Object,java.lang.Object>

└ java.util.Properties
└ **cvm.ucm.repository.Rep_Default_Properties**

All Implemented Interfaces:

java.io.Serializable, java.lang.Cloneable, java.util.Map<java.lang.Object,java.lang.Object>

```
public class Rep_Default_Properties
extends java.util.Properties
```

This class stores the default values for the system properties

Author:

Eduardo Monteiro

See Also:

Serialized Form

Field Summary

Fields inherited from class java.util.Properties
defaults

Constructor Summary

Rep_Default_Properties()

Method Summary

Methods inherited from class java.util.Properties
getProperty, getProperty, list, list, load, load, loadFromXML, propertyNames, save, setProperty, store, store, storeToXML, storeToXML, stringPropertyNames

Methods inherited from class java.util.Hashtable
clear, clone, contains, containsKey, containsValue, elements, entrySet, equals, get, hashCode, isEmpty, keys, keySet, put, putAll, rehash, remove, size, toString, values

Methods inherited from class java.lang.Object
finalize, getClass, notify, notifyAll, wait, wait, wait

Constructor Detail

Rep_Default_Properties

```
public Rep_Default_Properties()
```

---

cvm.ucm.repository

Class Rep_Properties

```
java.lang.Object
  └cvm.ucm.repository.Rep_Properties
```

```
public class Rep_Properties
extends java.lang.Object
```

This class stores the parameters required to create any type of repository.

Author:

Eduardo Monteiro

Field Summary

private java.io.File	**configFile**
private java.lang.String	**configFilename**
static java.lang.String	**DB_HOST**
static java.lang.String	**DB_NAME**
static java.lang.String	**DB_PASSWORD**
static java.lang.String	**DB_PORT**
static java.lang.String	**DB_USERNAME**
static java.lang.String	**FS_ROOT**
private java.util.Properties	**p**
static java.lang.String	**REP_TYPE**

Constructor Summary

Rep_Properties()

Method Summary

java.lang.String	**getProperty**(java.lang.String prop)
static void	**main**(java.lang.String[] args)
java.lang.String	**toString**()

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

DB_USERNAME

public static final java.lang.String **DB_USERNAME**

See Also:

Constant Field Values

DB_PASSWORD

public static final java.lang.String **DB_PASSWORD**

See Also:

Constant Field Values

DB_HOST

public static final java.lang.String **DB_HOST**

See Also:

Constant Field Values

DB_PORT

public static final java.lang.String **DB_PORT**

See Also:

Constant Field Values

DB_NAME

public static final java.lang.String **DB_NAME**

See Also:

Constant Field Values

REP_TYPE
public static final java.lang.String **REP_TYPE**

See Also:

Constant Field Values

FS_ROOT
public static final java.lang.String **FS_ROOT**

See Also:

Constant Field Values

p
private java.util.Properties **p**

configFilename
private java.lang.String **configFilename**

configFile
private java.io.File **configFile**

Constructor Detail

Rep_Properties
public **Rep_Properties**()

Method Detail

getProperty
public java.lang.String **getProperty**(java.lang.String prop)

toString
public java.lang.String **toString**()

Overrides:

toString in class java.lang.Object

main
public static void **main**(java.lang.String[] args)

cvm.ucm.repository

Class RepositoryType
java.lang.Object
  └ **cvm.ucm.repository.RepositoryType**

public abstract class **RepositoryType**
extends java.lang.Object

This class holds the definition of the different repository types.

Author:
Eduardo Monteiro

Field Summary

`static java.lang.String`	**`ACCESS`**
`static java.lang.String`	**`FILE_SYSTEM`**
`static java.lang.String`	**`MYSQL`**

Constructor Summary

`RepositoryType``()`

Method Summary

Methods inherited from class java.lang.Object
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Field Detail

ACCESS
`public static final java.lang.String` **`ACCESS`**

See Also:
Constant Field Values

MYSQL
`public static final java.lang.String` **`MYSQL`**

See Also:
Constant Field Values

FILE_SYSTEM
`public static final java.lang.String` **`FILE_SYSTEM`**

See Also:
Constant Field Values

Constructor Detail

RepositoryType

```
public RepositoryType()
```

---

cvm.ucm.repository

Class SourceDBLoader

```
java.lang.Object
  └ cvm.ucm.repository.SourceDBLoader
```

All Implemented Interfaces:

Sources

```
public class SourceDBLoader
extends java.lang.Object
implements Sources
```

This class retrieves a Macro from a database given a function name. There is no overloading of a function name. The schema for the table Macros, stored in the Repository db, is {name:string, returnType:string, paramTypeList:string, paramNameList:string, script:string}.

Field Summary

private static java.sql.Connection	**conn**
private static SourceDBLoader	**instance**
private static java.sql.ResultSet	**srs**
private static java.sql.Statement	**stmt**

Constructor Summary

private	**SourceDBLoader**(java.sql.Connection c) Creates a new instance of MacroLoader

Method Summary

static SourceDBLoader	**Instance**(java.sql.Connection c) * This is the Instance implementation of Singleton Design Pattern.
Macro	**loader**(java.lang.String name)

	Construct the Macro object from the information stored in the database
private java.util.ArrayList	**parser**(java.lang.String s) To parse a string into and ArrayList

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

conn
private static java.sql.Connection **conn**

---

stmt
private static java.sql.Statement **stmt**

---

srs
private static java.sql.ResultSet **srs**

---

instance
private static SourceDBLoader **instance**

Constructor Detail

SourceDBLoader
private **SourceDBLoader**(java.sql.Connection c)

Creates a new instance of MacroLoader

Method Detail

Instance
public static SourceDBLoader **Instance**(java.sql.Connection c)

* This is the Instance implementation of Singleton Design Pattern.

Parameters:

c - Connection DB object.

Returns:

SourceDBLoader instance.

---

parser
private java.util.ArrayList **parser**(java.lang.String s)

To parse a string into and ArrayList

Parameters:

s: - the string to be parsed

Returns:

list: the ArrayList with the contents of the string token by token.

loader

```
public Macro loader(java.lang.String name)
            throws java.lang.Exception
```

Construct the Macro object from the information stored in the database

Specified by:

`loader` in interface `Sources`

Parameters:

`name` : - the name of the function

Returns:

Macro: the Macro object for the given function.

Throws:

`java.lang.Exception`

See Also:

`Macro`

---

cvm.ucm.repository

Class SourceFileSysLoader

```
java.lang.Object
  └ cvm.ucm.repository.SourceFileSysLoader
```

All Implemented Interfaces:

Sources

```
public class SourceFileSysLoader
extends java.lang.Object
implements Sources
```

This class retrieves a Macro from a file system given a function name. There is no overloading of a function name. This has been created for future implementation availability.

Field Summary

private static SourceFileSysLoader	**instance**

Constructor Summary

SourceFileSysLoader()

Method Summary

static SourceFileSysLoader	**Instance**()
Macro	**loader**(java.lang.String name)

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

instance

private static SourceFileSysLoader **instance**

Constructor Detail

SourceFileSysLoader

public **SourceFileSysLoader**()

Method Detail

Instance

public static SourceFileSysLoader **Instance**()

---

loader

public Macro **loader**(java.lang.String name)

Specified by:

loader in interface Sources

---

cvm.ucm.repository

Class UCM_R_Facade

java.lang.Object

└ **cvm.ucm.repository.UCM_R_Facade**

---

public class **UCM_R_Facade**
extends java.lang.Object

Façade class that provides an interface into the UCM_Repository subsystem. The purpose of this class is to aid future expansion of this subsystem. OCL Statement: Context: UCM_R_Facade inv: self<>null

Author:

Frank Hernandez

Field Summary

private MacroLoader	**repLoad**

Constructor Summary

UCM_R_Facade()

Method Summary

Macro	**loadMacro**(java.lang.String name) This method creates a Macro object with the information obtained from the source for a given function name OCL Statement: Context: UCM_R_Facade::loadMacro(name) pre: name.length>0 Context: UCM_R_Facade::loadMacro(name) post: self.loadMacro(pre.name) <> null
static void	**main**(java.lang.String[] args) Subsystem Test - Repository Subsystem.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

repLoad

private MacroLoader **repLoad**

Constructor Detail

UCM_R_Facade

public **UCM_R_Facade**()

Method Detail

loadMacro

public Macro **loadMacro**(java.lang.String name)
throws java.lang.Exception

This method creates a Macro object with the information obtained from the source for a given function name OCL Statement: Context: UCM_R_Facade::loadMacro(name) pre: name.length>0 Context: UCM_R_Facade::loadMacro(name) post: self.loadMacro(pre.name) <> null

Parameters:

name: - the name of the function whose macro is required

Returns:
Macro object
Throws:
`java.lang.Exception`
See Also:
`Macro`

---

main

```
public static void main(java.lang.String[] args)
```

Subsystem Test - Repository Subsystem. Test Case ID: UCM_T_12 Unit Test - UCM_R_Facade Unit Test UCM_T_26

Parameters:
`args` -

10.6 Appendix F – Documented Code for Test Driver

cvm.se

Class SynthesisEngine

```
java.lang.Object
  └ cvm.se.SynthesisEngine
```

```
public class SynthesisEngine
extends java.lang.Object
```

This class is a simple implementation of the Synthesis Engine.

Author:
Frank Hernandez

Field Summary	
private static SynthesisEngine	**instance**

Constructor Summary	
private	**SynthesisEngine**()

Method Summary

static SynthesisEngine	**Instance**() Implementation of the instance method as in the Singleton Design Pattern.
static void	**main**(java.lang.String[] args) Driver for the SE.
static void	**notify**(Handles_Event event) This method handles the events reported to the SE.
static void	**resetSE**() Resets the SE.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

instance

private static SynthesisEngine **instance**

Constructor Detail

SynthesisEngine

private **SynthesisEngine**()

Method Detail

Instance

public static SynthesisEngine **Instance**()

Implementation of the instance method as in the Singleton Design Pattern.

Returns:

SynthesisEngine instance

---

resetSE

public static void **resetSE**()

Resets the SE.

---

notify

public static void **notify**(Handles_Event event)

This method handles the events reported to the SE.

Parameters:

event -

---

main

public static void **main**(java.lang.String[] args)

Driver for the SE. Runs Test Cases From ID: UCM_T_01 - UCM_T_14

Parameters:

```
args -
```

10.7 Appendix G – Diary of meeting and tasks.

Place:	ECS 212		
Date:	09/03/2007	**Facilitator:**	Dr.Peter Clarke
Start:	11:30AM	**Attending:**	Frank,Raidel, Guangqiang,Abhishek
End:	12:30AM	**Minute Taker:**	Abhishek

1. **Objective:**
   To get a brief idea of the CVM as a whole and also the specifics of our project. The functioning of the UCM subcomponent and its interface with the SE and NCB components.
2. **Status:**
   Got an idea of our project details where we will be working on the UCM engine and our users will be the SE and NCB subsystems. The discussions about the control scripts and the particular function calls were held and handouts with information are circulated.
3. **Discussion:**
   Had a discussion on the control scripts and the functions to be achieved by UCM. The NCB API's and the function calls from SE is collected and circulated through mail.
4. **Tasks:**
   It is decided that each project member will have to write 4 use cases**.**

Place:	VH 131		
Date:	09/05/2007	**Facilitator:**	Frank Hernandez
Start:	08:00PM	**Attending:**	Frank,Raidel,Guangqian Eduardo,Marylurdys, Abhishek
End:	09:00PM	**Minute Taker:**	Abhishek

1. **Objective:**
   To divide the use cases among the team members. 24 use cases are identified for the different scenarios in the system and distributed equally among the 6 project team members.
2. **Status:**
   Each team member is assigned 4 use cases and 2 misuse cases.
3. **Discussion:**
   The UCM architecture and its various functions are discussed. Also the different types of event handling and exceptions are discussed.
4. **Tasks:**
   All team members will write use cases and misuse cases before the next meeting.

Place:	VH 131		
Date:	09/12/2007	**Facilitator:**	Frank Hernandez
Start:	08:00PM	**Attending:**	Frank, Raidel, Eduardo, Marylurdysh, Guangqiang, Abhishek
End:	09:00PM	**Minute Taker:**	Abhishek

1. **Objective:**
   To plan the next phase of the Requirements Analysis. To work on the Analysis Models and come up with the UML diagrams according to the Use Cases developed.
2. **Status:**
   Presently, completed all the 24 use cases that are required. Each member contributed 4 use cases. 2 Misuse Cases need to be figured out and then can proceed with the next step for Object Diagrams, Sequence Diagrams of the selected Use Cases.
3. **Discussion:**
   Discussed on all the 24 Use Cases that are completed. Identified 8 use cases which are important and which are going to be implemented. Object Diagrams and Sequence Diagrams for the 8 Use Cases need to be worked on.2 Misuse Cases are also required to be developed. Also had discussions on meeting times and other types of communication means such as Skype among the team members.
4. **Tasks:**
   Object Diagrams and Sequence Diagrams are distributed and assigned for each team member.
   Need to complete UML diagrams before the next meeting. Next meeting scheduled on 09/16. Also need to work on Misuse cases.
   Every team member needs to join Google Groups for uploading stuffs and discussion on Project

Place:	ECS 212		
Date:	09/16/2007	**Facilitator:**	Frank Hernandez
Start:	03:00PM	**Attending:**	Frank, Guangqia, Marylurdys, Abhishek
End:	03:45PM	**Minute Taker:**	Abhishek

1. **Objective:**

   To have a discussion on the status of the progress and also on the UML diagrams for the 8 selected use cases.

2. **Status:**

   Present status is that the entire use cases are completed by all the team members.

3. **Discussion :**

   The SendSchema use case was selected for presentation in the next class. A brief ppt on the system architecture and the SendSchema use case is discussed.

   Had a discussion on the sequence diagrams for each team member. The Object Diagram was not clear and Frank will talk to Dr.Clarke regarding that.

   For use cases, values are required to be assigned to Scenarios. Also spelling errors need to be checked along with the fixing of the constraints in the use cases.

   Also had the requirement for Skype as means for virtual communication among the team members apart from the weekly meetings.

4. **Tasks:**

   Sequence Diagrams and Object Diagrams to be ready by all team members before next meeting. Sequence Diagrams to be uploaded to Google Groups by Monday.

   Use Cases to be fixed by every team member which are the Scenario values, spelling errors and the constraints.

   Mary will prepare and upload a sample PPT for the in-class presentation. She is also going to present her use case(Send Schema).

   All team members are asked to create a Skype account and post the respective ID's as soon as possible for better communication.

Place:	ECS 252		
Date:	09/23/2007	**Facilitator:**	Frank Hernandez
Start:	11:00AM	**Attending:**	Frank, Raidel, Eduardo, Marylurdys, Guangqiang, Abhishek
End:	01:00PM	**Minute Taker:**	Abhishek

1. **Objective:**

   To complete the Software Requirements Document(SRD) and checking the Use Cases of all the members for the final version of the deliverable. Also to develop some idea about the class design for the major components of the subsystem for the next phase.

2. **Status:**

   The Software Requirements Document (SRD) is completed with some minor additions and the entire document compiled for the final deliverable. All the 24 Use Cases form all the team members and 2 Misuse Cases along with the UML diagrams for the Use Case Models and Analysis Models (Static and Dynamic) are complete to be included in the SRD.

3. **Discussion:**

   All the 24 Use Cases were checked completely by each member for Scenarios, Constraints and errors in spelling. The Object Diagrams and the Sequence Diagrams for the final 8 Use Cases were also checked for correctness. Team members had a discussion on the basic software design plan with respect to Classes and the packages that are needed to realize the basic architectural plan of the UCM subsystem. Also discussed the interfaces that are required for communication with the NCB and the SE.

4. **Task**

   Checking the final document and if any modifications are necessary for the final deliverable of the SRD. Develop some ideas regarding the classes and packages that are required for communication among the subcomponents in the UCM. Once the required basic software design is ready then it will be assigned among the team members to work on individually for the final implementation of the system.

Place:	VH 131		
Date:	10/10/2007	**Facilitator:**	Marylurdys
Start:	08:30PM	**Attending:**	Frank,Raidel, Guangqiang,Abhishek, Eduardo
End:	09:00PM	**Minute Taker:**	Abhishek

1. **Objective:**
   To start working with the next phase i.e. Design Document. Identified the roles for each member in the team and reviewed the main requirements for this phase.
2. **Status:**
   The initial phase of Software Requirements Document(SRD) finished with delivering the document and a presentation to the client. Now our team is ready for the System Design and Object Design phases that we will be covering in this phase.
3. **Discussion:**
   Had a preliminary discussion on the basic system architecture and what types of architectural patterns to be used. Since we are required to use two patterns, so we have to identify two basic patterns specific for our application.
4. **Tasks:**
   All the team members were assigned to think and come up with ideas of architectural patterns for our system as well as the basic classes that may be required to develop our application. Also a brief idea of the subsystem decomposition in our system.

Place:	VH 131		
Date:	10/17/2007	**Facilitator:**	Marylurdys
Start:	08:00PM	**Attending:**	Frank,Raidel,Guangqian Eduardo, Abhishek
End:	09:00PM	**Minute Taker:**	Abhishek

1. **Objective:**

   To finalize on the major subsystems that will be required for our implementation part in the system. As well as to specify the interfaces for each subsystem and a brief discussion on the type of architectural patterns to be employed among our subsystems.

2. **Status:**

   We are started with our System Design phase, which mainly covers the Subsystem Decomposition and specifying interfaces for each subsystem. Also started to build our application code, which will help in identifying subsystems as well as architectural patterns required for our system.

3. **Discussion:**

   The main subsystems were discussed and reviewed. Also have shared some ideas of packaging the subsystems. The major subsystems identified are : Manager, Interpreter, Repository, Exception Handler, and EventHandler. The major functions of each subsystem were identified and shared some views on the implementation details of the system.

4. **Tasks:**

   Team members were assigned to review the Subsystem Decomposition and develop some basic ideas on the classes and objects that will be required for each subsystem. Also, the packaging of subsystems are required to be done.

Place:	ECS 252		
Date:	10/20/2007	**Facilitator:**	Marylurdys
Start:	11:00PM	**Attending:**	Frank, Raidel, Eduardo,Guangqiang, Abhishek
End:	02:00PM	**Minute Taker:**	Abhishek

1. **Objective:**
   To finalize the interfaces for the subsystem as well as the classes and methods required for each class. To identify two architectural patterns that will be implemented among the subsystems for our application.
2. **Status:**
   Presently, finished with our subsystem decomposition and identifying the general classes that are required. The detailed classes as well the methods and attributes for the classes as well the interfaces need to be done before starting up with the Object Design phase.
3. **Discussion:**
   Based on our Subsystem Decomposition we identified the requirement for two Architectural Patterns for our application, which will be Microkernel and Repository . Microkernel architecture will be implemented on our major component i.e. Interpreter and the Repository architecture will be implemented on our Database for loading and retrieving macros.
4. **Tasks:**
   Subsystems are divided among the team members so that we can start off with the implementation phase. Frank, Raidel and Eduardo will work on Interpreter. Maryludys and Abhishek will work on Repository. Guangqiang will be working on ExceptionHandler and EventHandler subsystem. Before the next meeting we need to have our subsystem interfaces ready and start off with the implementation.

Place:	VH 131		
Date:	10/24/2007	**Facilitator:**	Marylurdys
Start:	08:30 PM	**Attending:**	Frank, Guangqiang, Eduardo, Raidel, Abhishek
End:	09:00 PM	**Minute Taker:**	Abhishek

1. **Objective:**

   To have a discussion on our project status and getting started with the Object Design phase of our project. Also need to discuss on the design patterns that we are going to implement in our application for refinement of the system design and to identify if any additional objects are required for our application.

2. **Status:**

   Presently we have completed Subsystem Decomposition and the integration of our Architectural Patterns in our system. Also completed with the specification of subsystem interface and working on our implementation part.

3. **Discussion :**

   The main discussion was on the subsystem interfaces that were developed by the team members and whether any additional objects are required or not. Also shared views on the various Design Patterns that could be implemented in our classes.

4. **Tasks:**

   Each team member will carry on with its implementation for various subsystems. Also need to develop some idea on the possible design patterns that could be implemented for system realization.

Place:	ECS 252		
Date:	10/27/2007	**Facilitator:**	Marylurdys
Start:	01:00 PM	**Attending:**	Frank, Raidel, Eduardo,Guangqiang, Abhishek
End:	04:00 PM	**Minute Taker:**	Abhishek

1. **Objective:**

   To complete the Design Document and the various points that are required to be covered for the second deliverable. Reviewing the system architecture to identify the design patterns that could be employed in our system. To track the status for each implementation of subsystems.

2. **Status:**

   The System Design phase is completed and we are in midway through our Object Design phase. The subsystems were identified along with the architectural patterns and a basic implementation framework is under progress.

3. **Discussion:**

   We have decided on the four design patterns that are to be employed in our application as Façade, Command, Singleton and Strategy for the various classes in subsystems. The implementation classes should use this design patterns and work further. The database schema defined and was loaded with a few tuples for checking the implementation. The other parts of the application such as Parser, Interpreter, Exception Handler and Manger is also under construction

4. **Task :**

   The team members are required to submit the various classes from each subsystem along with the basic description for each classes. The subsystems should be ready and make to work completely as early as possible.

Place:	VH 131		
Date:	11/07/2007	**Facilitator:**	Eduardo
Start:	08:30PM	**Attending:**	Frank,Raidel, Guangqiang,Abhishek, Eduardo
End:	09:00PM	**Minute Taker:**	Abhishek

1. **Objective:**
   To start working with the next phase i.e. Testing Phase and the Final Deliverable. Identified the roles for each member in the team in this phase and reviewed the main requirements.
2. **Status:**
   The second phase of Design Document is already finished and delivered to the client. Now our team is ready to complete the final implementation part and then start off with the Testing phase of the system.
3. **Discussion:**
   We had a discussion on the final implementation details where we finalized that there will be three types of Repository for storing Macros. The three types will be: MS Access Database, MySQL Database and File System Organization. We decided on the fact that we are going to implement the MS Access and MySQL DB as repository and leave the File System Repository for future implementation but keeping the space for it in our current implementation.
4. **Tasks:**
   All the team members were given the deadline for finishing off their incomplete implementation parts and any modifications for the same so that by the next meeting we are completely done with the implementation. Also discussed a little bit on the Testing and how we are going to implement the Testing phase.

Place:	ECS 252		
Date:	11/10/2007	**Facilitator:**	Eduardo
Start:	01:00PM	**Attending:**	Frank, Guangqian Eduardo, Abhishek
End:	03:00PM	**Minute Taker:**	Abhishek

1. **Objective:**

   To complete the implementation part and start off with the Testing Phase.
   To discuss specific points on System Tests, Subsystem Tests and Unit Tests.
   To generate and distribute the Test Cases among all the team members.

2. **Status:**

   We are completed with the Implementation Phase and about to start off with the Testing Phase where we need to generate Test Cases for System Test, Subsystem Test and Unit Tests. We have also built up a general idea for our Testing phase which we are going to implement it by developing test drivers and stubs and then a final evaluation of all the Test Cases.

3. **Discussion:**

   The Testing plan was reviewed and discussed among the team members and there are some suggestions. We are required to write 24 Test Cases for System Tests and Subsystem Test (3 Test Cases for each of the 8 Use Cases that are implemented). Since our Use Cases are all the basic function calls to NCB or SE so we need to develop test drivers and test code so that all the function calls are tested and verified for its correct functioning.

4. **Tasks:**

   All the Test Cases are evenly distributed among the team members and also the necessary supporting codes that are required to execute the Test Cases for its correct functioning. Test Cases are numbered by an unique identifier which corresponds to the Use Case it tests and all the team members are required to complete the Test Cases along with the respective tests before the next meeting.

Place:	VH 131		
Date:	11/14/2007	**Facilitator:**	Eduardo
Start:	08:00PM	**Attending:**	Frank, Raidel, Eduardo,Guangqiang, Abhishek, Marylurdys
End:	09:00PM	**Minute Taker:**	Abhishek

1. **Objective:**

   To finalize and complete the Testing Phase where we are required to perform Unit Tests for the classes in the state machine and also to complete with the evaluation of all the Test Cases and complete the documentation for the Testing Phase.

2. **Status:**

   Presently completed with the System Tests, Subsystem Tests and also the Unit Test. The tasks to be finalized are the evaluation of the test results and the final documentation of the Test results as well as the Final Deliverable.

3. **Discussion:**

   Based on our state machine we have completed our Unit Test for the classes and also the System Tests and Subsystem Tests. We discussed on the final evaluation report of all the test cases and the documentation for the same. We also finalized on the various topics that are needed to cover for the Final Deliverable. The complete document for the final deliverable should be over as soon as possible.

4. **Tasks:**

   The remaining tasks in hand are: User Guide for our System, Compiled Meeting Diary for all the meetings in this project, Evaluation and complete documentation for the testing phase and the entire source code to be copied in a CD and the final compilation of all the required documents for the Final Deliverable. The Final Deliverable documents should be ready by the next weekend ready for submission after which we can prepare for the Final Presentation.