Collaborative design (COLLDESIGN): A real-time interactive unified modeling language tool

Surya Telikapalli

Follow this and additional works at: https://scholarworks.lib.csusb.edu/etd-project

Part of the Computer Sciences Commons

Recommended Citation
https://scholarworks.lib.csusb.edu/etd-project/2669

This Project is brought to you for free and open access by the John M. Pfau Library at CSUSB ScholarWorks. It has been accepted for inclusion in Theses Digitization Project by an authorized administrator of CSUSB ScholarWorks. For more information, please contact scholarworks@csusb.edu.
COLLABORATIVE DESIGN (COLLDESIGN): A REAL-TIME INTERACTIVE UNIFIED MODELING LANGUAGE TOOL

A Project
Presented to the
Faculty of
California State University,
San Bernardino

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
in
Computer Science

by
Surya Telikapalli
March 2004
COLLABORATIVE DESIGN (COLLDESIGN): A REAL-TIME INTERACTIVE
UNIFIED MODELING LANGUAGE TOOL

A Project
Presented to the
Faculty of
California State University,
San Bernardino

by
Surya Telikapalli
March 2004

Approved by:
Dr. A.I. Concepcion, Chair, Computer Science

Dr. George Georgiou

Prof. Ernesto Gomez

18 Mar 2004
ABSTRACT

Collaborative Computing is a very efficient way for physically dispersed team members to collaborate over common tasks over distance and time. Collaborative Computing offers significant advantages over single-user systems in that it brings together multiple perspectives and expertise, saves time and cost in coordinating group work, facilitates group problem-solving, and enables new modes of communication such as real-time interactive conferencing, distributed presentations etc. Since software design and modeling requires extensive communication and coordination between different team members, a collaborative software modeling tool can be of great help.

Collaborative Design (COLLDESIGN) is an interactive collaborative modeling tool. The initial version of COLLDESIGN was developed by Mr. Hara Totapally as part of Masters’ project under the guidance of Dr. A.I. Concepcion. The initial version included a collaborative framework comprising of configurable client and server components. The framework was extended to create COLLDESIGN supporting Unified Modeling Language limited to class diagram in part, in a collaborative environment. In the current version, a
complete implementation of the Class Diagram view has been accomplished. In addition, extending the framework, text messaging and audio conferencing features have been implemented to allow for real-time textual and audio communication between team members working on a particular project.
ACKNOWLEDGEMENTS

Sincere appreciation for the support provided to the project by the NSF MII grant.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>iii</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>v</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>xv</td>
</tr>
<tr>
<td>CHAPTER ONE: INTRODUCTION</td>
<td></td>
</tr>
<tr>
<td>Collaborative Computing</td>
<td>1</td>
</tr>
<tr>
<td>Need for a Collaborative Modeling Tool</td>
<td>3</td>
</tr>
<tr>
<td>About CollDesign</td>
<td>4</td>
</tr>
<tr>
<td>Summary</td>
<td>5</td>
</tr>
<tr>
<td>CHAPTER TWO: SOFTWARE REQUIREMENTS SPECIFICATION</td>
<td></td>
</tr>
<tr>
<td>Introduction</td>
<td>6</td>
</tr>
<tr>
<td>CollDesign Scope</td>
<td>6</td>
</tr>
<tr>
<td>Definitions, Acronyms, and Abbreviations</td>
<td>9</td>
</tr>
<tr>
<td>General Description</td>
<td>13</td>
</tr>
<tr>
<td>Product Perspective</td>
<td>13</td>
</tr>
<tr>
<td>Product Functions</td>
<td>14</td>
</tr>
<tr>
<td>User Characteristics</td>
<td>15</td>
</tr>
<tr>
<td>Constraints, Assumptions and Dependencies</td>
<td>15</td>
</tr>
<tr>
<td>Specific Requirements</td>
<td>16</td>
</tr>
<tr>
<td>Functional Requirements</td>
<td>16</td>
</tr>
<tr>
<td>TOC Pane</td>
<td>17</td>
</tr>
<tr>
<td>Logical View</td>
<td>17</td>
</tr>
</tbody>
</table>

vi
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maintainability</td>
<td>26</td>
</tr>
<tr>
<td>Portability</td>
<td>26</td>
</tr>
<tr>
<td>Summary</td>
<td>26</td>
</tr>
<tr>
<td><strong>CHAPTER THREE: SOFTWARE ARCHITECTURE</strong></td>
<td></td>
</tr>
<tr>
<td>Overview</td>
<td>27</td>
</tr>
<tr>
<td>Design</td>
<td>27</td>
</tr>
<tr>
<td>Project Design</td>
<td>27</td>
</tr>
<tr>
<td>Central Server</td>
<td>28</td>
</tr>
<tr>
<td>Distributed Client</td>
<td>28</td>
</tr>
<tr>
<td>CollDesign Packages</td>
<td>34</td>
</tr>
<tr>
<td>Request Façade</td>
<td>35</td>
</tr>
<tr>
<td>Response Façade</td>
<td>37</td>
</tr>
<tr>
<td>RequestPublisher Interface</td>
<td>38</td>
</tr>
<tr>
<td>RequestProcessor Façade</td>
<td>39</td>
</tr>
<tr>
<td>RequestProcessorFactory</td>
<td>40</td>
</tr>
<tr>
<td>IResponsePublisher</td>
<td>41</td>
</tr>
<tr>
<td>DefaultRequestProcessor</td>
<td>42</td>
</tr>
<tr>
<td>RequestValidator</td>
<td>43</td>
</tr>
<tr>
<td>Client</td>
<td>45</td>
</tr>
<tr>
<td>RequestDispatcher</td>
<td>47</td>
</tr>
<tr>
<td>Connecting to Server</td>
<td>49</td>
</tr>
<tr>
<td>Dispatching Request</td>
<td>50</td>
</tr>
</tbody>
</table>
ClientListener ................................. 51
Establishing Connection with the Server ...... 51
Receiving Response from Server ............... 52
Creating Requests ............................... 52
Response Processing .............................. 53
Audio Messaging ................................. 56
Creating an Audio Request(LOCK_AUDIO) ...... 60
Server Processing the Audio Request
(LOCK_AUDIO) ................................ 61
Response Processing on the Client-Side ...... 62
Text Messaging ................................. 63
Creating a Chat Request ...................... 63
Server Processing the Chat Request .......... 64
Response Processing on the Client-Side ...... 65
Adding a Relationship ........................... 67
Creating Classdiagram Request .............. 68
Server Processing the Classdiagram Request .. 69
Response Processing on the Client-Side ...... 70
Summary ........................................ 72

CHAPTER FOUR: TESTING

Overview ........................................ 73
Test Cases ....................................... 73
Class Requests .................................. 73
Create a New Class.............................. 73
Checkout a Class............................. 74
Checkin a Class............................... 75
Rename a Class................................. 76
Delete a Class................................. 77
Add New Attribute............................. 78
Add an Operation............................... 79
Attribute Requests............................. 80
   Rename an Attribute......................... 80
   Delete an Attribute.......................... 81
Operation Requests............................ 82
   Rename an Operation.......................... 82
   Delete an Operation........................... 83
Classdiagram Requests.......................... 84
   Create a New Classdiagram.................... 85
   Delete a Classdiagram......................... 85
   Rename a Classdiagram......................... 86
   Checkout a Classdiagram....................... 88
   Checkin a Classdiagram......................... 89
   Add a Class................................ 90
   Add Association Relationship............... 92
   Add Inheritance Relationship................. 93
Add Aggregation Relationship............. 94
Add Dependency Relationship.............. 97
Multiple Class Checkout.................... 98
Multiple Class Checkin..................... 100
Audio Messaging ................................ 101
Text Messaging ................................ 102
Summary ........................................ 105

CHAPTER FIVE: MAINTENANCE MANUAL

Overview ....................................... 106
Directory Structure ............................ 106
Bin .............................................. 107
Configuration .................................... 107
Java ............................................. 107
Html ............................................ 108
Client Configuration ........................... 108
Client Initialization ........................... 108
Port ............................................ 108
BindPort ....................................... 108
BindName ....................................... 108
MainMenu ...................................... 109
MainToolBar .................................... 109
ClassDiagToolBar .............................. 109
ResponseProcessors ............................ 109
Tool .................................................. 109
Server Configuration ............................. 109
Serverinit.prop ................................. 110
Application Properties ....................... 110
  Title ............................................. 110
  Stdout ......................................... 110
  Stderr ......................................... 110
ServerSocket Properties ..................... 110
  Port ............................................ 110
VideoRMIServer Properties ................ 111
  BindName ...................................... 111
  BindAddress .................................. 111
  BindPort ...................................... 111
RequestProcessors.prop ..................... 111
Build Environment ............................ 111
  Setenv.bat .................................... 112
Build.properties .............................. 112
  Port .......................................... 112
  BindPort ...................................... 112
  BindAddress .................................. 112
  Webserver .................................... 112

xii
Java Home: ................................. 112
Build.xml ................................. 113
Project .................................... 113
Name ....................................... 113
Default ................................... 113
Basedir ................................. 113
Properties ................................. 113
Src ......................................... 113
Config .................................... 114
Html ....................................... 114
Bin ......................................... 114
Dist ....................................... 114
Lib ......................................... 114
Client .................................... 114
Server .................................... 114
Webroot ................................... 114
Targets .................................... 114
Init ....................................... 115
Compile ................................. 115
Rmic ..................................... 115
Client .................................... 115
Deploy .................................... 116
CHAPTER SIX: CONCLUSIONS AND FUTURE DIRECTIONS

Overview ............................................. 118
Future Directions ...................................... 120
  Authentication and Authorization ............. 120
  Load Balancing and Session Management ...... 120
  Support for Non-Web Applications ........... 121
  Using SVG .......................................... 121
  User to User Communication .................... 121
Summary .................................................. 121

REFERENCES .......................................... 122
LIST OF FIGURES

Figure 1. Graphical User Interface .................................. 17
Figure 2. The Collaborative Framework .............................. 29
Figure 3. Class Request/Response .................................... 30
Figure 4. Classdiagram Request/Response ............................ 31
Figure 5. Attribute/Operation Request/Response ..................... 31
Figure 6. CollDesign State Diagram .................................. 33
Figure 7. CollDesign Architecture ................................. 34
Figure 8. CollDesign Packages ............................... 35
Figure 9. Interface IRequest ........................................ 36
Figure 10. Request Façade ........................................ 36
Figure 11. Interface IResponse ...................................... 37
Figure 12. Response Façade .......................................... 38
Figure 13. IRequestPublisher Interface ............................. 39
Figure 14. IRequestProcessorInterface ............................. 39
Figure 15. RequestProcessor Façade ............................... 40
Figure 16. RequestProcessorFactory ............................... 41
Figure 17. IResponsePublisher ................................. 41
Figure 18. DefaultRequestProcessor ............................... 43
Figure 19. Request Validation ...................................... 44
Figure 20. CollDesign Client Composition .......................... 46
Figure 21. RequestDispatcherFactory ............................... 48
Figure 22. RequestDispatcher Composition .................. 49
Figure 23. Connecting to Server .............................. 50
Figure 24. Request Dispatching ............................... 51
Figure 25. Event Notification Chain ......................... 53
Figure 26. Response Publishing by ClientListener .......... 54
Figure 27. IResponseProcessor ............................... 55
Figure 28. Response Processor Façade ....................... 55
Figure 29. Audio Control Panel .............................. 58
Figure 30. Dispatching Audio Request ....................... 60
Figure 31. Processing Audio Request ......................... 61
Figure 32. Processing Audio Response ....................... 63
Figure 33. Requesting Chat Event ........................... 64
Figure 34. Processing Chat Request .......................... 65
Figure 35. Processing Chat Response ....................... 67
Figure 36. Requesting a Relationship ....................... 69
Figure 37. Processing Relation Request ..................... 70
Figure 38. Processing Relationship Response ............... 72
Figure 39. Create New Class ................................. 74
Figure 40. Checkout a Class ................................... 75
Figure 41. Checkin a Class .................................... 76
Figure 42. Renaming a Class .................................. 77
Figure 43. Deleting a Class .................................... 78

xvi
Figure 44. Add New Attribute ................................. 79
Figure 45. Adding a New Operation .............................. 80
Figure 46. Renaming an Attribute ................................. 81
Figure 47. Deleting an Attribute ................................. 82
Figure 48. Renaming an Operation ................................. 83
Figure 49. Delete an Operation ................................. 84
Figure 50. Creating a New Classdiagram ......................... 85
Figure 51. Deleting a Classdiagram .............................. 86
Figure 52. Renaming a Classdiagram .............................. 87
Figure 53. Checkout a Classdiagram .............................. 88
Figure 54. Checkin a Classdiagram .............................. 89
Figure 55. Add Class Tool ......................................... 91
Figure 56. Classdiagram Manager .............................. 92
Figure 57. Add Association Relation ......................... 93
Figure 58. Add Inheritance Relation ............................. 94
Figure 59. Add Aggregation Relation ......................... 96
Figure 60. Multiplicity Selection Window ...................... 96
Figure 61. Add Dependency Relation ......................... 98
Figure 62. Selecting Area to Perform Multiple Checkout ................................................. 99
Figure 63. Multiple Class Checkout ............................. 100
Figure 64. Multiple Class Checkin ............................. 101
Figure 65. Audio Control Panel .............................. 102
Figure 66. Starting Text Messaging ..................... 104
Figure 67. Chat Window .................................. 105
CHAPTER ONE

INTRODUCTION

Collaborative Computing

Collaborative Computing provides a very efficient way in which physically dispersed team members can collaborate over common tasks over distance and/or time. The technology offers significant advantages over single-user systems in that it brings together multiple perspectives and expertise, saves time and cost in coordinating group work, facilitates group problem-solving, and enables new modes of communication such as real-time interactive conferencing, distributed presentations etc., Following are some of the educational and commercial software packages that have been successfully implemented using collaborative technologies:

- CCF (Collaborative Computing Frameworks) a multidisciplinary research project at Emory University provides a suite of software systems, communications protocols, and tools that enable collaborative, computer-based cooperative work. CCF constructs a virtual work environment on multiple computer systems connected over the Internet, to form a Collaboratory. [16]
• PCCE (The Pervasive Collaborative Computing Environment) developed by Distributed Systems department of Berkeley labs allows geographically dispersed collaborators to use collaborative features such as sending instant messages, holding private and group conversations, sharing documents, tracking current work flow and dynamically activating application sharing and videoconferencing sessions as needed.

• TUTOS (The Ultimate Team Organization Software) is an open source tool to manage organizational needs of small groups, teams or departments. TUTOS provides web-based tools such as user calendar, an address manager, bug tracker, project repository, mailboxes and project time tracker. [17]

• DISCIPLE (Distributed System for Collaborative Information Processing and Learning) developed by Center for Advanced Information Processing at Rutgers University enables participants at different locations to collaboratively access, manipulate, analyze, and evaluate multimedia data. [18]
• SunForum 3.2 'Data Collaboration' software supports audio- and video-conferencing and whiteboard sessions, VoIP telephony communication, share applications, and transfer files and multimedia information.[20]

• Kansas a computing environment developed by Sun Microsystems Laboratories' Interactive Collaborative Systems group enables real-time collaboration between participating users. The system features video, audio, and a flexible, programmable, object oriented desktop environment.

Need for a Collaborative Modeling Tool

Software modeling is an essential component of any software system lifecycle. A good software model acts as a medium of communication between different project team members and ensures a sound architecture. UML (Unified Modeling Language) has become an industrial standard language for software modeling. Several modeling tools supporting UML specifications are available in the market. Rational Rose, TogetherJ, Visio etc. are some of them. Most of these tools are single-user desktop applications.

A software system development lifecycle involves active participation of people with diverse expertise from
Business Analysts, Software Analysts, Programmers, and Quality Assurance Engineers to Documentation Writers. A software system’s success therefore heavily depends on extensive communication and collaboration between these diverse team members. A collaborative modeling tool can be of great help in the process.

With this vision, the initial version of CollDesign was developed. A flexible client/server collaborative framework was implemented. Using the framework, a partial implementation of class diagram view was then developed. In the current version, complete implementation of the class diagram view has been accomplished. In addition, text messaging and audio conferencing features have been added to function as communication mediums between different team members.

About CollDesign

CollDesign (Collaborative Design Tool) is an object-oriented collaborative software design tool. CollDesign allows software designers to graphically create UML software designs. The chief purpose of CollDesign is to allow software designers located at geographically distinct locations to get involved in designing a project and being
able to observe modifications or new functionalities made by other users real-time. The initial version of CollDesign implemented a flexible framework for client/server communication and a partial implementation of the class diagram view in adherence with Unified Modeling Language (UML) specifications. The goal of the current version was to completely implement the class diagram view and to allow different software designers working on a particular project at a given time to interact with each other via audio conferencing and text messaging, all this using the framework of the initial version.

Summary

In this chapter, advantages of collaborative computing are described. A need for a collaborative design tool is discussed. A brief introduction to CollDesign is provided.
CHAPTER TWO
SOFTWARE REQUIREMENTS SPECIFICATION

This document attempts to capture the functional requirements of Collaborative Design Tool (CollDesign). This document describes the purpose, function and design of CollDesign.

Introduction
CollDesign is a collaborative UML modeling tool that allows multiple software designers to work on a single Object Oriented model. It enables software designers located at geographically distinct locations to participate in designing class diagram view of a project and also observe modifications or new functionalities made by other users real-time. The system also allows these multiple users to communicate with each other via text messaging and audio conferencing.

CollDesign Scope
Since the Unified Modeling Language (UML) specification is very large it is beyond the scope of this project to implement all of its features.
Listed below are the features available in CollDesign project:

Class. The following operations related to a class are supported:

- Create a new Class
- Checkout a Class
- Check in a Class
- Rename a Class
- Delete a Class
- Add an Attribute to a Class
- Add an Operation to the Class

Attribute. The following operations related to an attribute are supported:

- Rename an Attribute
- Delete an Attribute from a Class

Operation. The following operations related to an operation are supported:

- Rename an Operation
- Delete an Operation from a Class

Class diagram. The following operations related to a class diagram are supported:
• Create a new Class diagram
• Delete a Class diagram
• Rename a Class diagram
• Check out a Class diagram
• Add a Class to the Class diagram
• Check in a Class diagram
• Delete a Class from the Class diagram

Association. The following operations related to association relationship are supported
  • Create an association relation
  • Check out an association relation
  • Modify an association relation
  • Delete association relation

Inheritance. The following operations related to inheritance relationship are supported
  • Create an inheritance relation
  • Check out an Inheritance relation
  • Delete inheritance relation

Aggregation. The following operations related to aggregation relationship are supported
  • Create an aggregation relation
• Check out an aggregation relation
• Modify an aggregation relation
• Associate multiplicity
• Delete aggregation relation

Dependency. The following operations related to a dependency relationship are supported

• Create a Dependency relation
• Check out a Dependency relation
• Delete Dependency Relation

Multiple Check in/Checkout. The operations related to multiple Checkin/Checkout are supported.

• Check out multiple classes from the class diagram view.

• Check in multiple classes from the class diagram view

Text Messaging. Allows users to communicate with other users via text conversation.

Audio Messaging. Allows users to communicate with other users via audio conversation.

Definitions, Acronyms, and Abbreviations

The definitions, acronyms, and abbreviations used in the document are described in this section:
SRS: Software Requirements Specifications

CollDesign: Collaborative Design Tool

Applet: An applet is a small program that is intended not to be run on its own, but rather to be embedded inside another application.

Abstract Factory: Provides an interface for creating families of related or dependent objects without specifying their concrete classes.

Builder: Separates the construction of a complex object from its representation so that the same construction process can create different representations.

Chain of Responsibility: Avoiding coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.

Command: Encapsulates a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.

Composite: Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual and compositions of objects uniformly.
Decorator: Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to sub-classing for extending functionality.

Façade: Provide a unified interface to a set of interfaces in a subsystem. Façade defines a higher-level interface that makes the subsystem easier to use.

Factory Method: Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.

Flyweight: Use sharing to support large numbers of fine-grained objects efficiently.

Iterator: Provides a way to access the elements of an aggregate object sequentially without exposing its underlying representation.

Mediator: Defines an object that encapsulates how a set of objects interact. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and lets you vary their interaction independently.

Naming: The naming class provides methods for storing and obtaining references to remote objects in the remote object registry.
Observer: Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.
Singleton: Ensure a class only has one instance and provide global point of access to it.
Persistency: Ensure a class only has one instance and provide global point of access to it.
Java: Persistence indicates that a collection of data remains intact even if its source is no longer attached to the network.
Serialization: Java is a cross-platform programming language from Sun Microsystems.
Server Socket: Serialization is the process of writing/reading the persistent data to/from the storage media.
Socket: A server socket waits for requests to come in over the network. It performs some operation, and then possibly returns a result to the requester.
Swing: Swing is the part of the Java Foundation Classes (JFC) that implements a new set of GUI components with a pluggable look and feel.
TCP/IP: Transmission Control Protocol on top of the
Internet Protocol provides a reliable, point-to-point communication channel those client-server applications on the internet use to communicate with each other.

UML: The Unified Modeling Language is an open method used to specify, visualize, construct and document the artifacts of an object-oriented software-intensive system under development. The UML represents a compilation of best engineering practices which have proven successful in modeling large, complex systems.

Web Server: A web server is an application, which would serve contents to the World Wide Web.

XML: XML is an abbreviation for extensible markup language.

General Description

This section contains a description of the overall product and its requirements. It provides a perspective for understanding the specific requirements from specific requirements section of this chapter.

Product Perspective

CollDesign has been developed using Object Oriented methodology. The Class Diagram view of CollDesign has been developed in adherence to Unified Modeling Language specifications.
To ensure that CollDesign works across platforms it has been developed in 100% Java. CollDesign has been developed as a web based application that can be run over local area network or over the internet.

**Product Functions**

CollDesign provides an applet with an easy to use Graphical user Interface developed using Swing components. CollDesign enables software designers possibly situated at distinct locations to design the class diagram view of a software system in adherence to UML specifications. The system allows the users to create, edit and modify classes, attributes, operations and class diagrams. To ensure synchronicity a check in/checkout mechanism has been implemented. In addition, the system also allows establishing relationships (association, aggregation, dependency and inheritance) between different classes in a class diagram view. A user can check out a relationship and modify it as well. Also a user can communicate with other users logged onto the system via audio conferencing or text messaging. A detailed functionality of CollDesign is described in the Functional Requirements section of this chapter.
User Characteristics

Users are expected to be Software designers located possibly at distinct locations. The users are expected to have good knowledge and experience using modeling tools such as Rational Rose, Visio etc.

Constraints, Assumptions and Dependencies

The Development language followed is Java (JDK 1.4). The application will be running in JRE 1.4.

- The User should have good knowledge and experience using software modeling tools, preferably Rational Rose.
- The client’s minimum hardware requirements for CollDesign are Pentium III processor, 256 MB Memory and 2 MB of available disk space.
- CollDesign requires a web-browser, either Internet Explorer 5.5+ or Netscape Navigator 6+
- CollDesign assumes that JRE 1.4 is installed on the client machine.
- CollDesign assumes that the web browser used is enabled to display swing components.
- Since the application has been developed in 100% Java, it can be run under most platforms, Windows
98/NT/2000/XP, UNIX and Linux.

- CollDesign being a web-based application requires a local area network and preferably internet as well.

Specific Requirements

This section contains all the details which are relevant for the design phase to follow. The ordering given here is just one way to present the specific requirements in a logical way.

Functional Requirements

CollDesign provides as the Graphical user interface, an applet developed using Swing Components. The Interface will have a table of contents pane to the left side of the applet, a toolbar containing all the tools available in the application to the center and a design pane where the classes and relationships are displayed to the right.
Figure 1. Graphical User Interface

TOC Pane. The Table of Contents pane displays the different views available in the application such as the logical view (the class diagram view) and the Chat View.

Logical View.

- A user intending to use the logical view should first select the logical view node in the TOC Pane.
- Upon right clicking on the logical view, a pop-up menu
appears from which a user can add a class or a class diagram.

- To add a class, the user should click on the class menu item. A class node appears below the logical view node with a default name of “New Class”.

- To rename the class, a user has to first check out the class to ensure synchronicity of the application for all the clients. To checkout a class the user must right click on the class node. By selecting the checkout menu item the user checks out the class, meaning no other user can modify the class properties. The class node icon will now display in red.

- The user can then rename the class by again right clicking on the class node and selecting the rename menu item. A dialog requesting the new name pops up.

- The user has to enter the new intended name for the class and press the OK button. The class node now has a new name.

- To add an attribute or an operation to the class the user must right click on the class node and choose the attribute or the operation menu item based on which class property the user intends to add. Now an
attribute/operation node appears below the class node.

- The user can rename the attributes or the operations in the same way as for the class node.

- To add a class diagram to the view, the user must right click on the logical view node and choose the class diagram menu item from the pop-up menu.

- To rename or check out the class diagram the same procedure as described for the class node has to be followed.

- In addition, to display the class diagram in the design pane, a user must double click on the class diagram node.

- To check in a class or a class diagram node, the user can right click on the particular node and select the check in option from the Menu item. This means that the user is giving away the lock from the node. A different user can now checkout the node and do modifications.

- At any time to delete any of the nodes, the user can right click on the particular node and select the delete option from the popup menu. The node will then be removed from the application.
• While the user is doing these operations, all the other distributed clients are refreshed real-time displaying the changes made by the user.

**Chat View.** A user can communicate with other users via text messaging through the Chat View. To start the communication a user must right click on the chat view and choose the start option from the pop up menu. A Chat dialog then pops up. The user must first set a screen name before sending messages to other users. The text field to enter the screen name is available in the top pane of the dialog. Messages typed in the message text field will be broadcasted to all other users currently using CollDesign. For the other users to view these messages they must have their chat dialogs open as well.

**Main Toolbar.** The Main Toolbar is situated in the top pane of the applet. A user intending to communicate with other users via audio messaging can use the VCR type buttons available in the Main Toolbar. Firstly, the user has to make sure that a microphone is available and connected to the workstation. To record a message to be sent to other users, the user must first click on the Record button available in the Main toolbar. The user must
then speak the message into the microphone. The application, records this message on the local disk of the user machine. Upon completing the message, the user must click on the send button. The application then broadcasts the recorded message to all other users. To listen to the message, the users on the other side should have a microphone or at least a speaker connected to their machine.

**ClassDiagram Toolbar.** The toolbar is located at the center of the applet separating the TOC Pane and the Design pane. All the tools available in the application reside in this toolbar.

**Class.** To display a class in the design pane the following steps have to be followed in the same order:

- A user has to first double click on a class diagram node from the TOC Pane. A class diagram window opens up in the design pane.
- To display a class in the class diagram window, the class diagram node in the TOC pane has to be checked out.
- The user must then click at a location in the class diagram window where the class has to be displayed and
then click on the class tool button in the toolbar. The class will then show up on the class diagram window. All the attributes and the operations associated with the class are also displayed in the class model.

- Renaming an attribute or operation associated with that class from the TOC Pane is reflected in the class diagram window of all the clients.

- Any new attribute or operation added to the class node in the TOC pane is also reflected in the class model.

**Relationships.** The toolbar has tool buttons to add association, inheritance, dependency or aggregation relationships between two class models in the class diagram window.

To add a relationship the following steps have to be followed:

- Both the classes to which a relationship has be added have to be checked out from the TOC Pane first.

- The user must then click on the tool button associated to that relationship in the toolbar.

- Clicking on a location in close vicinity of the “from class” in the class diagram window, the user must drag
the mouse until the line drawn touches the "to class".

- A dialog pops up requesting the user to choose the multiplicity between the two classes. Upon selecting the multiplicity and pressing the ok button, the relationship with the appropriate representation is displayed between the two classes. The multiplicity is also displayed.

**Design Pane.** The design pane is located to the right side of the applet. The design pane is where all the display happens. Along with displaying classes and relationships between them driven from the toolbar, a user can also do the following operations from the Design Pane:

- A user can do a multiple class checkout by selecting an area in the design pane such that the classes fall within the boundaries of the selection. The user must then right click within the selection boundaries, a pop up menu appears. By selecting the multiple check out option, the user can checkout all the classes as long as none of them are already checked out by some other user.

- In the same way, a multiple check in can be done from the design pane.
• A user can also checkout a particular relationship between two classes and modify it.

• While all these operations are happening, the synchronicity of the application across the distributed clients is maintained by the central server.

External Interfaces

System Interfaces. The system can run both as an intranet as well as an internet solution.

User Interfaces. The system provides a graphical user interface (GUI) to enable users to create, modify and access classes, operations, attributes, class diagrams and relationships. In addition, the GUI enables a user to communicate with other users via text messaging and audio conferencing. The GUI is basically an applet developed using Java Swing components.

Hardware Interfaces. There are no specific hardware interfaces in this project. Hardware control is generally handled by drivers and other sound players for that particular platform.

Software Interfaces. Software interfaces made by the system will be in use.
Communications Interfaces. The system uses sockets and RMI, which are TCP/IP connections over a network; the network could be a local area network or the Internet. Audio data will be sent raw over these links.

Memory Constraints. Each workstation must have a minimum 256 MB of RAM to run CollDesign. The client system should take negligible disk space unless when audio is being written to disk.

Design Constraints

Operating System Constraint. The system should run on most platforms, Windows 98/2000/NT/XP, Linux and UNIX.

GUI Constraint. In case of using a web browser to run the system, the browser has to be enabled to display Java Swing components.

Programming Language Constraint. The system has been developed using Java 1.4.

Project Deliverables Constraint. The system and the documentation will be made available on a Compact Disc.

Software System Attributes

Reliability. CollDesign is expected to reliably accept the requests from the distributed clients and publish responses and maintain synchronicity between the clients.
Availability. Unless for maintenance purposes or for upgrades, CollDesign may be made available at all times.

Security. Security is not a design issue at this time. An authorization process can be put in place at a later stage.

Maintainability. The system has been developed in 100% Java adhering to Java 1.4 specifications utilizing design patterns. The system should therefore be easy to maintain.

Portability. Since the system has been developed totally in Java, a multi-platform language, it is expected to work across platforms.

Summary

The software requirements specifications for CollDesign are discussed in this chapter. In the next chapter CollDesign architecture is explained in detail and the implementation details of various components are discussed.
CHAPTER THREE
SOFTWARE ARCHITECTURE

Overview

In this chapter CollDesign architecture, CollDesign package structure, CollDesign components, and use case realization with the help of sequence diagrams are explained.

Design

Project Design

Taking advantage of design patterns and web technology a collaborative framework was developed in the initial version of CollDesign. The framework was designed as a client/server technology by employing publisher/subscriber design pattern. A client sends a request to the central server, the server processes the request and publishes the response. Since all distributed clients currently connected are set up as response subscribers, they receive the response thus ensuring synchronicity of the application. Extending the framework, CollDesign, a collaborative modeling tool has been implemented.
Central Server

The central server is responsible for request processing and response publishing. The server was implemented using a combination of Remote Method Invocation (RMI) and sockets. RMI queues the requests from the clients and the ServerSocket maintains the list of the connected clients to broadcast the response.

Distributed Client

The distributed client was designed and developed as an applet using swing components for the graphical user interface. The client can be run on any web browser or as an application using java appletviewer.
The use case scenario for class request or response is shown in Figure 3.
Figure 3. Class Request/Response

The use case scenario for class diagram request or response is shown in Figure 4.
The use case scenario for attribute/operation request or response is shown in Figure 5.

Figure 4. Class Diagram Request/Response

Figure 5. Attribute/Operation Request/Response
CollDesign is based on the client/server architecture. Communication between the clients is established through a centralized server. The server component of CollDesign has been developed in a multi-threaded multi-client scenario. The server component uses Remote Method Invocation (RMI) to receive incoming requests. To broadcast responses to the distributed clients, the server component uses TCP/IP Sockets. Each client connected to the centralized server creates and dispatches requests to the server through a Graphical User Interface.

The server upon receiving a request performs the following operations in the same order:

- Adds the request to a queue
- Sorts the request queue
- Dequeues the request queue,
- Instantiates a request processor
- Generates a response after processing the request
- Logs the response and finally
- Broadcasts the response.

Each of the distributed clients receives the response and processes it thus ensuring a synchronicity across the application. This complete cycle of request processing and
response publishing is graphically represented as a state diagram in Figure 6.
CollDesign Packages

CollDesign source code has been developed in the following package structure:

video: All the classes commonly used by both the client and the server are located in this package.

server: All the classes utilized by the central server reside in this package.

client: All the classes used by the distributed clients including the graphical interface are located in this package.

event: All the classes which perform event handling jobs.
reside in this package. Contains events used in the application.

utilities: All the utility classes are located in this package.

drawingtools: drawing tool components such as straight line etc., reside in this package.

Figure 8. CollDesign Packages

Request Façade

Facade design pattern is employed to send requests from the client to the server. IRequest is the unified interface based on which concrete request classes are implemented. In addition to being the base interface, IRequest extends Comparator interface to sort the queued requests. The sorting is done based on the initial timestamp. Each request has mutator functions to get initial time and the InetAddress of the request source.
class Comparator

interface IRequest

+String GetComponentName()
+long getIntTime()
+void setIntTime(long initTime)
+InetAddress getSource()
+void setSource(InetAddress inetAddress)

Figure 9. Interface IRequest

Figure 10. Request Façade
Response Façade

For every request in the request Façade there is a corresponding response class in the response façade. Similar to the request process, the response mechanism too uses Façade design pattern. The base interface for the response Façade is IResponse. IResponse interface and the response Façade are shown in figure 11 and Figure 12.

```
interface video.IResponse

+String getComponentName()
```

Figure 11. Interface IResponse
For publishing the incoming requests IRequestPublisher interface is defined. IRequestPublisher has three abstract methods to be implemented by all concrete request publisher classes. publishRequest method is defined to publish requests. addRequestSubscriber method is defined to add subscribers. removeSubscriber method is defined to remove any subscribers associated with the RequestPublisher. The interface IRequestPublisher is shown in Figure 13.
interface
server.IRequestPublisher

| +void publishRequest(final IRequestPublisher requestPublisher, final IRequest request) |
| +void publishRequest(final IRequestPublisher requestPublisher, final IRequest request, final InetAddress inetAddress) |
| +void addRequestSubscriber(IRequestSubscriber requestSubscriber) |
| +void removeRequestSubscriber(IRequestSubscriber requestSubscriber) |

Figure 13. IRequestPublisher Interface

RequestProcessor Façade

For request processing, façade design pattern is employed. The base interface defined for request processing is IRequestProcessor. The interface has one abstract method, processRequest. IRequestProcessor Interface and the request processor Façade are shown in Figure 14 and Figure 15.

interface
server.IRequestProcessor

+IResponse processRequest(IRequest request)

Figure 14. IRequestProcessorInterface
The interface for creating a request processor is defined by the factory method IRequestProcessorFactory Method and RequestProcessorFactory class implements the IRequestProcessorFactoryMethod. The RequestProcessor Factory class diagram is shown in Figure 16.
Figure 16. RequestProcessorFactory

IResponsePublisher

To publish response, IResponsePublisher interface is defined. The interface has methods to add and remove subscribers. For publishing response, publishResponse method is defined. The IResponsePublisher interface is shown in Figure 17.

Figure 17. IResponsePublisher
DefaultRequestProcessor

DefaultRequestProcessor is a multi-threaded component. It enqueues incoming requests in one thread. In another thread it dequeues the requests and delegates request processing responsibility to a dynamic request processor. For each type of a request a corresponding RequestProcessor is instantiated dynamically. It uses RequestProcessors.prop configuration file for this purpose. The type of request can is obtained using request.getClass().getName() method. Since each request is mapped to a request processor in the configuration file, the corresponding request processor is instantiated Class.forName(request processor name).newInstance(). Since all RequestProcessors should implement IRequestProcessor interface, the request processor loaded is cast to IRequestProcessor. In addition to implementing IRequestProcessor, DefaultRequestProcessor class also implements IResponsePublisher. It implements the publishResponse method to publish the response returned by the RequestProcessor.
Every request is subjected to validation by RequestValidator. The validation process for each type of request is discussed below:

LockRequest: LockRegistry maintains the locks of the different objects of the application. It is the responsibility of the LockRegistry to check if a lock has already been issued on an object. Validation is applied to a lock request with RELEASE action to allow only the request source to release a lock.

AttributeRequest: Every AttributeRequest is subjected to
the test that the parent class was locked and the request was from the client which has the lock.

OperationRequest: Every OperationRequest is subjected to the test that the parent class was locked and the request was from the client which has the lock.

ClassRequest: Creating a new class does not require a validation. Any other related request is validated for the class being locked and the request is from the same client, which possesses the lock.

ClassdiagramRequest: Creation of a new Classdiagram is not validated. Any other related request is validated for a lock and the requesting client’s identity.

ChatRequest: No validation is done for a chat request.

AudioRequest: No validation is done for an audio request.

Figure 19. Request Validation
When a client requests a connection close, the socket is removed from the connected client list. All the locks requested by the disconnecting client are removed from the LockRegistry.

Client

CollDesign Client was implemented as an Applet supporting swing components. The responsibilities of the client side of the application include:

• connecting with server
• Sending requests to the server
• Listening for responses from the server
• Processing responses from the server

The client has the following main components to meet the above requirements:

VideoClient: VideoClient is the GUI of the application
RequestDispatcher: RequestDispatcher is responsible for dispatching requests to the server.
ClientListener: ClientListener is responsible for listening to the incoming responses from the server.
Registry: Registry is the repository for the application. The Registry would be a mirror image of the registry on the server.
LockRegistry: Maintains a list of locks of the objects in the application and the corresponding source.

WindowManager: WindowManager is responsible for managing various windows of the application.

VideoTOC: VideoTOC is a tree display of the different components of the application.

StatusBar: StatusBar displays the status of the client.

The client was designed using all the above described components. The client's composition is shown in Figure 20.

Figure 20. CollDesign Client Composition
RequestDispatcher

A request dispatcher implemented as a singleton is responsible for dispatching requests to the server. The RequestDispatcher implements IRequestDispatcher interface for dispatching the requests. It extends the Observable class so that all of its observers can be notified of its state change. In addition, it should implement the interface IRequestEventListener so that it could be registered as a RequestEventListener of VideoClient.

AbstractRequestDispatcher is an implementation of the RequestDispatcher. RequestDispatcherFactory implements the interface IRequestDispatcherFactory which defines the interface for creating a request dispatcher. RequestDispatcherFactory class diagram is shown in Figure 21.
The RequestDispatcher implemented in the project uses Remote Method Invocation. The RequestDispatcher has reference to the VideoRMIServer. RequestDispatcher’s composition is shown in Figure 22.
Connecting to Server

The client connects to the server by looking up the VideoRMIServer on the network. The hostname is determined from the applet's getCodebase().getHost() method.
VideoRMIServer's configuration parameters bindName and bindPort are read from "client/clientinit.prop" file on the server. The process of establishing the connection with the server is shown in Figure 23.

![Sequence diagram](image)

Figure 23. Connecting to Server

**Dispatching Request**

RequestDispatcher dispatches the request to the server using VideoRMIServer.

Sequence diagram for dispatching a request is shown in Figure 24.
ClientListener

ClientListener is implemented as an Observable multithreaded object. StatusBar is registered as an Observer of ClientListener so that ClientListener's state change can be displayed on the statusbar. ClientListener creates a socket and connects to the server. ClientListener listens for the incoming responses from the server.

Establishing Connection with the Server

A connection to the server is established by ClientListener, which opens a TCP/IP socket and connects to the server.
Receiving Response from Server

ClientListener constantly listens for server's broadcast in its runnable implementation.

Creating Requests

The classes capable of generating a request event are:

- Attribute Node
- Operation Node
- Class Node
- Classdiagram Node
- LogicalViewNode
- ChatViewNode

"Chain of Responsibility" design pattern is used to channel all requests through the RequestDispatcher class. The event notification chain is shown in Figure 25.
Figure 25. Event Notification Chain

Response Processing

Every response object from the server is received by ClientListener first. ClientListener publishes the response and the registered response subscribers would process the response further. Response publishing by ClientListener and the collaboration diagram showing the response processing are shown in Figure 26.
As a registered response processor of ClientListener, VideoClient processes the response by invoking a dynamic response processor. The properties file "client/config/ResponseProcessors.prop" contains the mapping between ResponseClass and the ResponseProcessor to be dynamically invoked. Again for the dynamic invocation of the response processor the response processor façade is used. The interface IResponseProcessor defining the base interface and the response processor Façade are shown in Figure 27 and Figure 28. After creating the response processor Registry is registered as a response subscriber of IResponseProcessor.
Figure 27. IResponseProcessor

Figure 28. Response Processor Façade
Upon receiving a notification from the response processor Registry synchronizes the application state based on the response received and fires a new ResponseEvent based on the response it received. VideoTOC and WindowManager as registered response event listeners would receive the ResponseEvent. While WindowManager processes a ClassdiagramResponseEvent VideoTOC delegates the response event processing to LogicalViewNode. LogicalViewNode processes the ClassdiagramResponseEvent and ClassResponseEvent and delegates AttributeResponseEvent and OperationResponseEvent processing to ClassNode. ChatResponse Event is processed by ChatViewNode.

Audio Messaging

The Audio feature of CollDesign has been implemented to allow distributed clients using the tool to participate in audio communication. The feature has been designed utilizing the underlying framework. A client creates an audio request meaning that a client records an audio message. The message is then dispatched to the centralized server. The server upon receiving the request creates an Audio response. This audio response is broadcasted to all the clients connected to the system.
To integrate audio with the underlying framework, the feature uses the following classes.

- client\audio\IAudioPlayer.java
- client\audio\AudioPlayer.java
- client\audio\IAudioRecorder.java
- client\audio\AudioRecorder.java
- client\audio\IAudioRecorderListener.java

The main tool bar in the GUI part of the application has an audio control panel. There are basically two buttons, a "record" button to record an audio message and a "transmit" button to transmit the audio message to the server. The transmit button creates an audio request and dispatches it to the centralized server by means of a request dispatcher. The Figure 29 displays the audio control panel.
displayed on the button, a tooltip element that defines the message to be shown when a mouse is placed on the button and finally a command element which defines what function in the applet code should be invoked upon clicking the button.

```xml
<Button>
  <d><image>Record16.gif</image><d></d>
  <d><tooltip>Start Recording</tooltip><d></d>
  <d><command>Audio.TALK</command><d></d>
</Button>

<Button>
  <d><image>Stop16.gif</image><d></d>
  <d><tooltip>Stop Recording</tooltip><d></d>
  <d><command>Audio.STOP</command><d></d>
</Button>
```

VideoClient class creates the main toolbar by making use of a Toolbar Builder which parses the configuration file.

The ToolbarBuilder takes ActionListener and the toolbar as arguments in its constructor. The action events are chained to the VideoClient class which in turn implements the ActionListener interface.
Creating an Audio Request (LOCK_AUDIO)

Creation of an audio request involves the following steps:

- A user must first click on the audio recording button.
- An action event is fired as a result.
- This action event generated by the button is received by the VideoClient class.
- Based on the cmd associated with the button clicked, the VideoClient class then creates an audio request as shown below.
- It then dispatches the request to the server through the Request Dispatcher.

![Diagram of Audio Request Dispatching](image)

Figure 30. Dispatching Audio Request
Server Processing the Audio Request(LOCK_AUDIO)

For every audio request received by the server a corresponding request processor is created by the DefaultRequestProcessor.

Before an AudioRequest can be processed an AudioRequestProcessor needs to be created and mapped in server/config/RequestProcessors.prop file as shown below:

```
video.AudioRequest=server.AudioRequestProcessor
```

Since Façade pattern has been used in the application, the AudioRequestProcessor must implement IRequestProcessor interface.

The Audio response created by the Audio Request Processor is broadcasted to all the connected clients by the DefaultRequestProcessor class as follows:

```
publishResponse(requestProcessor.processRequest(request));
```

![Diagram](image)

Figure 31. Processing Audio Request
Response Processing on the Client-Side

The audio response is first received by VideoClient class which implements the IResponseProcessor interface. Any response received by the client is first handled in the method processResponse.

Façade pattern is employed on the client-side for response processing. For every response received, a dynamic response processor is created as defined in the “client/config/ResponseProcessors.prop” configuration file.

For processing an AudioResponse, AudioResponseProcessor class is created as one of the members of VideoClient class. This AudioResponseProcessor is registered as an AudioResponseProcessorListener. The processAudioResponseEvent method is responsible for handling the audio response based on the cmd associated with the event. If the cmd is TransmitData, then the AudioResponseProcessor plays the audio data received via the audio response.
Text Messaging

The text messaging feature of CollDesign is implemented to allow distributed clients using the tool to participate in textual communication. The feature has been designed utilizing the underlying framework. A client creates a Chat Request meaning that a client types in a text message. The message is then dispatched to the centralized server when the send button is clicked. The server upon receiving the request creates a Chat response. This chat response is broadcasted to all the clients connected to the system.

Creating a Chat Request

Creation of chat request involves the following steps: A user must initially right click on the Chat View Node and select the “Start” item.

Figure 32. Processing Audio Response
• The user must then enter a screen name.
• By entering a message and clicking on the send button, an action event is fired.
• This action event generated by the send button is received by the VideoTOC class.
• The VideoTOC class then creates a chat request.
• The request is then dispatched to the server through the Request Dispatcher.

![Diagram](image)

Figure 33. Requesting Chat Event

Server Processing the Chat Request

For every chat request received by the server a corresponding request processor is created by the DefaultRequestProcessor.

Before a chat request can be processed a ChatRequestProcessor needs to be created and mapped in
server/config/RequestProcessors.prop file as shown below:

```
video.ChatRequest=server.ChatRequestProcessor
```

Since Façade pattern has been used in the application, the ChatRequestProcessor must implement IRequestProcessor interface. The actual request processing is done in IRequestProcessor’s processRequest method.

The chat response created by the ChatRequestProcessor is broadcasted to all the connected clients by the DefaultRequestProcessor class as follows:

```
PublishResponse(requestProcessor.processRequest(request));
```

![Figure 34. Processing Chat Request](image)

**Response Processing on the Client-Side**

The chat response is first received by VideoClient class which implements the IResponseProcessor interface.
Any response received by the client is first handled in the method processResponse. Similar to the way the request processing is handled by the server, Façade pattern is employed on the client-side for response processing.

For every response received a dynamic response processor is created as defined in the "client/config/ResponseProcessors.prop" configuration file.

For processing a ChatResponse, ChatResponseProcessor class is created as one of the members of VideoClient class. This ChatResponseProcessor’s processResponse method sends the response event to all the subscribers to the IChatResponseListener. VideoTOC which is a registered listener receives the Chat Response event and delegates the responsibility of Response processing to ChatViewNode. The ChatViewNode receives the Chat Response and displays the message on the ChatFrame.
Adding a Relationship

CollDesign allows users to add relationships between classes. The relationships supported by CollDesign are Association, Inheritance, Aggregation and dependency. A client generates a ClassDiagramRequest setting the action to be to Add the desired relationship meaning that a client selects the corresponding relationship tool from ClassDiagram toolbar and does a mouse drag operation between a “from class” and a “to class” between whom the relation is to be established. The request is then dispatched to the centralized server. The server upon receiving the request creates a ClassDiagramResponse with the action being set to add the desired relationship. This response is broadcasted to all the clients connected to the
system. It is the responsibility of each of the connected clients to render the relationship in the design pane.

Creating Classdiagram Request

The actions associated with the four relationships supported by CollDesign are ADD_RELATION_EXTENDS, ADD_RELATION_AGGREGATION, ADD_RELATION_ASSOCIATION and ADD_RELATION_DEPENDENCY.

Creation of classdiagram request for one of these actions involves the following steps:

- A user must first click on the appropriate relationship tool from the ClassDiagram toolbar.
- It has to be ensured that the two classes between which the relationship is to be established are checked out by the user.
- The user must then do a mouse drag operation between the "From Class" and the "To Class" in the ClassDiagram displayed in the DesignPane. The action event is then fired
- This action event is then received by the ClassDiagramFrame class.
- The ClassDiagramFrame class instantiates a LockRegistry, checks if the classes involved are
checked out by the user and then creates a ClassDiagram request.

- The request is then dispatched to the server through the Request Dispatcher.

Figure 36. Requesting a Relationship

Server Processing the ClassDiagram Request

For every ClassDiagram request received by the server a corresponding request processor is created by the DefaultRequestProcessor.

Before a ClassDiagramRequest can be processed a ClassDiagramRequestProcessor needs to be created and mapped in server/config/RequestProcessors.prop file as shown below:

```java
video.ClassDiagramRequest =
    server.ClassDiagramRequestProcessor;
```
Since Façade pattern has been used in the application, the ClassDiagramRequestProcessor must implement IRequestProcessor interface. The actual request processing is done in IRequestProcessor's processRequest method.

The ClassDiagram response created by the ClassDiagramRequestProcessor is broadcasted to all the connected clients by the DefaultRequestProcessor class as follows:

PublishResponse(requestProcessor.processRequest(request));

![Diagram](image)

Figure 37. Processing Relation Request

Response Processing on the Client-Side

The ClassDiagram response is first received by VideoClient class which implements the IResponseProcessor interface. Any response received by the client is first handled in the method processResponse.
Similar to the way the request processing is handled by the server, Façade pattern is employed on the client-side for response processing.

For every response received a dynamic response processor is created as defined in the "client/config/ResponseProcessors.prop" configuration file.

For processing a ClassDiagramResponse, ClassDiagramResponseProcessor class is created as one of the members of VideoClient class. This ClassDiagramResponseProcessor's processResponse method sends the response event to all the subscribers to the IClassDiagramResponseListener. VideoTOC which is a registered listener receives the ClassDiagram Response event and delegates the responsibility of Response processing to LogicalViewNode. The LogicalViewNode receives the ClassDiagram Response and delegates the rendering responsibility to ClassDiagramRenderer. The ClassDiagramRenderer on the basis of the action associated with the ClassDiagram response delegates the final rendering responsibility to one of the following renderers:

- RelationExtendsRenderer
- RelationAggregationRenderer
• RelationAssociationRender

• RelationDependencyRenderer.

The appropriate renderer then handles the responsibility of displaying the relationship between the two selected classes.

Figure 38. Processing Relationship Response

Summary

In this chapter CollDesign architecture, CollDesign package structure, CollDesign components, and use case realization with the help of sequence diagrams were explained. In the next chapter test cases explaining the supported functionality is discussed.
CHAPTER FOUR

TESTING

Overview

In this chapter, test cases for all the functionalities within the scope of the project are explained. Steps to be followed for each test case are described. Snapshots for each test case are provided.

Test Cases

Test cases developed to demonstrate all the functionalities of CollDesign are described below:

Class Requests

All requests related to a class are created by using the class node's context menu except for creating a new class which is done through the logical view node's context menu.

Create a New Class. A new class is created from the logical view node's context menu. Creating a new class is shown below and the steps required to create a new class are:

• From the TOC select Logical View node
• Right click to pop up the context menu
• Select New->Class

Figure 39. Create New Class

Checkout a Class. Checking out a class is shown below.

• Select the class node required to check out
• Right click to pop up the context menu
• Select Checkout
Checkin a Class. The process of checking in a class is shown below. The steps involved in checkin a class are:

- Select the class node to check in
- Right click to bring up the context menu
- Select Check In
Figure 41. Checkin a Class

**Rename a Class.** The process of renaming a class is shown below. The steps required to rename a class are:

- Select the class node to rename
- Right click to popup the context menu
- Select Rename
- A dialog box prompting to enter the new class name would be displayed. Enter the new class name and select OK to rename or Cancel to cancel the operation.
Figure 42. Renaming a Class

Delete a Class. The process of deleting a class is shown below. The necessary steps to delete a class are:

- Select the class node
- Right click to popup the context menu
- Select delete
Figure 43. Deleting a Class

**Add New Attribute.** The process of adding a new class is shown below. The steps to add a new attribute are:

- Select the class node
- Right click to popup the context menu
- Select New->Attribute
Figure 44. Add New Attribute

Add an Operation. The process of adding a new operation is shown below. The steps to add a new operation are:

- Select the class node
- Right click to popup the context menu
- Select New->Operation
Figure 45. Adding a New Operation

Attribute Requests

Attribute requests are generated by using attribute node's context menu.

Rename an Attribute. The process of renaming an attribute is shown below. The steps to rename an attribute are:

- Select the attribute node to be renamed
- Right click to popup the context menu
- Select Rename
Figure 46. Renaming an Attribute

Delete an Attribute. The process of deleting an attribute is shown below. The necessary steps to delete an attribute are:

- Select the attribute node
- Right click to popup the context menu
- Select delete
Figure 47. Deleting an Attribute

Operation Requests

Operation requests are generated by using the operation node's context menu.

Rename an Operation. The process of renaming an operation is shown below. The steps to rename an operation are:

- Select the operation node to be renamed
- Right click to popup the context menu
- Select rename.
• A dialog box prompting to enter the new name for the attribute would be displayed. Enter the new name and select OK to rename the operation or select Cancel to cancel the renaming operation.

Figure 48. Renaming an Operation

Delete an Operation. The process of deleting an operation is shown below. The steps required to delete an operation are:

• Select the operation node to be deleted
- Right click to popup the context menu
- Select Delete

Figure 49. Delete an Operation

Classdiagram Requests

Request for creating a class diagram is generated from logical view node's context menu. Checkout, check-in, rename, and delete requests are generated from the classdiagram node's context menu.
Create a New Classdiagram. The process of creating a new Classdiagram is shown below. The steps required to create a new Classdiagram are:

- Select the logical view node
- Right click to popup the context menu
- Select New->Class Diagram

![Classdiagram creation](image)

Figure 50. Creating a New Classdiagram

Delete a Classdiagram. The process of deleting a Classdiagram is shown below. The detailed steps to delete
a classdiagram are:

- Select the classdiagram node that needs to be deleted
- Right click to popup the context menu
- Select Delete

Figure 51. Deleting a Classdiagram

Rename a Classdiagram. Renaming a classdiagram is shown below. The detailed steps to rename a classdiagram are:

- Select the classdiagram to be renamed
• Right click to popup the context menu
• Select Rename
• A dialog box prompting to enter the new name for the classdiagram is displayed.
• Enter the new name for the classdiagram and select OK to rename the classdiagram or Cancel to cancel the classdiagram renaming.

Figure 52. Renaming a Classdiagram
Checkout a Classdiagram. Checking out a classdiagram is shown below. The steps required to checking out a class diagram are:

- Select the classdiagram node to be checked out
- Right click to popup the context menu
- Select Checkout

Figure 53. Checkout a Classdiagram
Checkin a Classdiagram. Checking in a classdiagram is shown below. The steps required to checkin a classdiagram are:

• Select the classdiagram to be checked in

• Right click to popup the context menu

• Select Check In

![Diagram showing steps to check in a classdiagram]

Figure 54. Checkin a Classdiagram
Add a Class. A class to a class diagram is added interactively. First the tool for adding a class should be selected. The tool for adding a class to an active class diagram is shown below. After selecting the tool click on the class diagram window to add a class at the location. The class diagram manager dialog would be displayed prompting the user to select a class to add to the class diagram. The class diagram manager is shown below. From the class diagram manager select the classes to be added to the class diagram and select Apply to add the classes or Cancel to cancel the operation.
Figure 55. Add Class Tool
Figure 56. Class diagram Manager

**Add Association Relationship.** An association relationship between two classes can be interactively added to the class diagram. Firstly, it has to be ensured that the classes between which the association relationship is to be established are checked out. Then the tool to add association relationship should be selected. The mouse button has to be pressed in close proximity of the “From class” and dragged so that it touches the “To class”. On releasing the mouse, a dialog box pops up requesting to select multiplicity. The relationship is then displayed with the appropriate multiplicity.
Add Inheritance Relationship. An inheritance relationship between two classes can be interactively added to the class diagram. Firstly, it has to be ensured that the classes between which the inheritance relationship is to be established are checked out. The tool to add inheritance relationship should then be selected. The mouse button has to be pressed in close proximity of the "From
class and dragged so that it touches the "To class". On
releasing the mouse, a dialog box pops up requesting to
select multiplicity. The relationship is then displayed
with the appropriate multiplicity.

Figure 58. Add Inheritance Relation

Add Aggregation Relationship. An aggregation
relationship between two classes can be interactively added
to the class diagram. Firstly, it has to be ensured that the classes between which the aggregation relationship is to be established are checked out. The tool to add aggregation relationship should then be selected. The mouse button has to be pressed in close proximity of the "From class and dragged so that it touches the "To class". On releasing the mouse, a dialog box pops up requesting to select multiplicity. The relationship is then displayed with the appropriate multiplicity.
Figure 59. Add Aggregation Relation

Figure 60. Multiplicity Selection Window
Add Dependency Relationship. A dependency relationship between two classes can be interactively added to the class diagram. Firstly, it has to be ensured that the classes between which the dependency relationship is to be established are checked out. The tool to add dependency relationship should then be selected. The mouse button has to be pressed in close proximity of the "From class and dragged so that it touches the "To class". On releasing the mouse, a dependency relation is displayed between the two classes.
Multiple Class Checkout. Multiple classes can be checked out simultaneously from the class diagram. First it has to be made sure that none of the classes intended to be checked out are already checked out by other users. The tool to do multiple class checkout should be selected. A rectangular area has to be selected encompassing the classes to be checked out. On right clicking, a popup menu
appears. Selecting checkout option, checks out all the classes within the area.

Figure 62. Selecting Area to Perform Multiple Checkout
Multiple Class Checkin. Multiple classes can be checked in simultaneously from the class diagram. First it has to be made sure that all the classes intended to be checked in are already checked out by the user. The tool to do multiple class checkout should then be selected. A rectangular area has to be selected encompassing the classes to be checked in. On right clicking, a popup menu appears. Selecting checkin option, checks in all the classes within the area.
Figure 64. Multiple Class Checkin

Audio Messaging

An audio communication can be established between connected users with this feature. To use this feature, the user must first have a microphone connected to the workstation. The main tool bar has an audio control panel with two buttons. To record a message to be broadcasted to all other connected clients, the record button has to be first clicked. Then a message has to be spoken into the
microphone. Once the message is finished, the transmit button has to be clicked. The message will be broadcasted to all clients. The users having a microphone or speakers of their own can listen to the message.

Figure 65. Audio Control Panel

Text Messaging

A textual communication can be established between connected users with this feature. The Tab The main tool
bar has an audio control panel with two buttons. To record a message the Contents pane has a chat view. To send a message to all connected users, the right click mouse button has to be clicked on the Chat View. A popup menu is then displayed. Upon selecting the start menu, a chat window is displayed. A username has to be first selected before sending a message for the first time. To transmit the message to all other users, the transmit button has to then be clicked. All connected users with their chat window active can view the message.
Figure 66. Starting Text Messaging
Figure 67. Chat Window

Summary

In this chapter various test cases were explained with the help of screenshots. In the next chapter the build environment and maintenance manual are described.
Overview

In this chapter the directory structure employed in the application is described. The server and client configuration files are described. The build environment is described. Also configuring the application in a web environment and starting the server is described.

Directory Structure

The following directory structure has been employed for CollDesign.

- Images: Contains all the images used in the application
- Lib: Contains all the XML configuration files used in the application
- Src: Contains all the source code, binary files required for running the application, configuration files and the html files used in the application.
- Bin: Contains all the batch files used in the application.
- Configuration: Contains all the server and client
configuration files.

- **Java**: Contains all the java source code

**Html**: Contains all the html files used in the application.

**Bin**

The bin directory contains the batch files to run the client, the server and registry.

**Configuration**

The configuration directory contains two subdirectories client, and server. Each of these directories contains config directory and the client and sever configuration files are located in these directories.

**Java**

The java directory contains all the java source code of the application. The source code has been broadly divided into 7 packages: server, client, video, event, utilities and geom.

- **Server**: Contains all classes related to server
- **Client**: Contains all classes related to client
- **Video**: Contains all classes common to video
- **Event**: Contains all event classes used by the client
- **Drawintools**: Contains all classes used for client side drawing
- Utilities: Contains all utility classes
- Geom: contains all the geometry classes.

**Html**
Contains all the html files used in the application.

**Client Configuration**
Configuration on the client-side includes customization of the menu, customization of the different toolbars, response processor customization, images and tools. Described below are the various client side configuration files used in the application.

**Client Initialization**
"clientinit.prop" contains the list of all the properties used for client initialization in the form of key value pairs. The following properties are used:

- **Port.** This property defines the server socket port at which the server listens for incoming requests.

- **BindPort.** This property defines the port at which the rmiregistry listens.

- **BindName.** This property is used by the clients to do look ups. It describes the bound name of VideoServerImpl.

"images.prop" lists the key value pair objects and the corresponding images used in the application.
MainMenu

"MainMenu.xml" is used for the construction of client's menu.

MainToolBar

"MainToolBar.xml" is used for the construction of client's toolbar.

ClassDiagToolBar

"ClassDiagToolBar.xml" is used for the construction of Class diagram toolbar.

ResponseProcessors

"ResponseProcessors.prop" lists as key value pairs, the response object and the corresponding response processor to be invoked.

Tool

"tools.prop" lists the key value pairs of the commands and the associated invoked tools.

Server Configuration

The responsibility of the server component is to queue incoming client requests, managing the status of the clients and publishing responses to the clients. Server configuration is done using a properties file "serverinit.prop". For request processor customization, the
server uses a properties file "requestprocessors.prop".

**Serverinit.prop**

All the application properties, ServerSocket Properties and VideoRMIServer properties are defined in serverinit.prop.

**Application Properties**

The Application properties described in this file are the title of the application, standard output redirection and standard error redirection.

**Title.** This property defines the title of the application.

**Stdout.** This property defines the file name to which the standard output stream is redirected.

**Stderr.** This property defines the file name to which the standard error stream is redirected.

**ServerSocket Properties**

The server socket properties defined in the file are described below:

**Port.** This property defines the port number to which the serversocket is bound and where it listens to client connections. This is the same port number the serversocket uses to broadcast the response from the server.
VideoRMIServer Properties

VideoRMIServer is the component of the server which queues incoming requests from the clients. The VideoRMIServer properties defined in the file are described below.

**BindName.** This property enables the network to identify the VideoRMIServer. The clients use this same bind name to do the lookup process.

**BindAddress.** The rmiregistry is bound to an IP Address to ensure that there is no issue when the server has multiple network cards. This property defines that IPAddress.

**BindPort.** To assign a port number other the default 1099 to the RMIREgistry the bindPort property is used.

**RequestProcessors.prop**

"RequestProcessors.prop" lists request object and corresponding request processor as key-value pairs.

Build Environment

Ant is used for compiling the source code. The file "build.xml" is used for building the application. The compiled classes would be placed in the destination directory “build”, specified in the build.xml file.
Setenv.bat

This batch file is used to configure the Java Home and Ant Home of the server machine. Once paths for both the environment variables are set in the classpath, the Ant build file can be run from the command prompt.

Build.properties

This properties file is used to define the following application properties:

Port. This property defines the port number to which the serversocket is bound and where it listens to client connections. This is the same port number the serversocket uses to broadcast the response from the server.

BindPort. This property is used to define the port number at which Rmiregistry listens.

BindAddress. The rmiregistry is bound to an IPAddress to ensure that there is no issue when the server has multiple network cards. This property defines that IPAddress.

Webserver. This property defines the name of the webserver under which the application resides.

Java Home. This property defines the path of java home.
Build.xml

To automate compilation and deployment of an application, Apache Ant uses "build.xml" configuration file. Directory names, File names, Classpath and other important properties used at several places in the target tree are described using the property tags. The compilation and deployment is automated by using a target tree where various tasks get executed. Each task is run by an object that implements a particular Task interface. Described below are the various properties and targets defined in build.xml.

Project

All build files start with a project tag. All the properties and targets are included within the project tag. It has three attributes.

Name. It defines the name of the project
Default. It defines the default target to be executed
Basedir. It defines the base directory.

Properties

Following are the properties defined in build.xml:

Src. It defines the path of java source code relative to the base directory
Config. It defines the path of configuration directory

Html. It defines the path of the html directory.

Bin. It defines the path of the binary directory.

Dist. It defines the path of the distribution directory.

Lib. It defines the path of the library directory.

Client. It defines the path of the client distribution directory.

Server. It defines the path of the server distribution directory.

Webroot. It defines the path of the webserver under which the application is run.

Targets

Following is the target tree used to execute various tasks in automating the build process.

• Init: creates the build, and dist directory structures.

• Compile: compiles all java code into the build directory

• Rmic: runs rmic to generate the stub and skeletons for VideoServerImpl class.

• Client: generates the code necessary to deploy
the client. Copies the client configuration files

- **Server**: generates the code necessary to deploy the server. Copies the server configuration files
- **Deploy**: deploys the client to the webserver
- **Clean**: deletes the destination and distribution directories for a fresh compile.

**Init**. This target is used to create build, client distribution, server distribution and logs directories.

**Compile**. This target is used to compile all the java source code used in the application to the build directory.

**Rmic**. This target runs rmic to generate the stub and skeletons for VideoServerImpl class.

**Client**. This target first creates the client distribution files. It then copies the client side java classes in the build directory to the client distribution directory. It copies client side xml and prop configuration files to client\config directory. Finally it substitutes the properties in the client configuration files with the property values in build.properties. During this process the values specified in build.properties are substituted in the client configuration files. The files in
dist/client_dist are required to be deployed to the webserver.

**Deploy.** This target deploys the client distribution files to the webserver. The webserver root directory is obtained from build.properties.

**Server.** Creates the distribution files for server. It then substitutes the properties in the server configuration files with the property values in build.properties. The files in dist/server_dist are used to run the server.

**Clean.** This target is used to delete the destination and distribution directories for a fresh compile.

**Web Server Setup**

The setting up of server consists of copying class files, images and html files to the web server. The class files are placed in a web client accessible directory.

**Web Server Configuration**

Copy all class files preserving the package structure to the web server's directory from where the application would be served.

**Starting Server**

"server.bat" and "RemoteServer.bat" starts the server and the rmiregistry respectively.
Connecting to Server

"client.bat" batch file invokes the appletviewer and connects to the server.

Summary

In this chapter the directory structure of the application was described. Client and server configuration files and details were described. The build environment has been described.

Web server set up and the batch files to start the server, and connecting to the server were described.
CHAPTER SIX
CONCLUSIONS AND FUTURE DIRECTIONS

Overview

CollDesign being a collaborative software modeling tool requires client server architecture. The communication between the client and server was established by making use of a combination of Remote Method Invocation and sockets. To ease scalability of the application a flexible collaborative framework was implemented in the initial version. The current version involves extension of features of CollDesign making use of the underlying framework.

The initial version of CollDesign included a partial implementation of the classdiagram view. In the current version, the classdiagram view was fully implemented by allowing association, aggregation, generalization, and dependency relationships to be established between classes. In addition, multiple checkin and checkout of classes from the design pane has been implemented. Harnessing the power of collaborative computing, audio messaging and text messaging features have been implemented.
Features implemented in this version of the project are enumerated below:

- Establishing association, aggregation, generalization, and dependency relationships between classes belonging to a class diagram.
- Modifying association and aggregation relations between classes.
- Deleting Relations between two classes
- Multiple Class Checkin and Checkout of classes from a class diagram
- Text messaging, allowing users to communicate via chat.
- Audio messaging, allowing users to communicate via audio conversation.

The front end of the application is an applet supporting swing components. The look and feel of the applet is kept similar to that of Rational Rose for users to easily transition to CollDesign.

The collaborative framework allows for easy scalability for future development. All that is needed is a new type of a request, a corresponding response class, a request processor on the server side to process the request,
and a responseprocessor on the client side to accept the response from the server. The framework being generic, it can be easily extended to develop new collaborative applications as well.

The robustness and reliability of the application can be improved by introducing some enterprise features to the collaborative framework. Features such as authorization and authentication, load balancing, multiple application support and user to user communication will add to security as well as dependability of the application.

Future Directions

Authentication and Authorization

To ensure that only authorized users are allowed to use the application an authorization and authentication mechanism can be added to the framework. LDAP (Light Weight Directory Access Protocol) is a very commonly used one such mechanism.

Load Balancing and Session Management

The current framework supports a distributed clients/centralized server architecture. One of the major drawbacks of such architecture is that it has a single point of failure. Instead, if the framework is modified to allow
multiple servers hosting the application and synchronizing them through session management, the reliability can be improved.

Support for Non-Web Applications

The framework could be modified to support standalone applications in addition to the applets.

Using SVG

Applets being fat clients require more download time. Instead using thin clients such as Scalar Vector Graphics (SVG) can enhance the performance of the application.

User to User Communication

The framework could be modified to allow a user to interact with a particular user of his choice instead of broadcasting the messages to all the connected users.

Summary

In this chapter, a brief overview of the application has been discussed. Some suggestions for future directions have been put forward.
REFERENCES

[1]. "IEEE Recommended Practice for Software Requirements Specifications (IEEE Std. 830-1993)"


[4]. Sun Microsystems, "Java Platform 1.4 Specification"


[12]. Collaborative Virtual Workspace (CVW), http://cvw.sourceforge.net/


