REUSABLE MODULES FOR BUILDING GRAPHICAL USER INTERFACES UNDER MOTIF AND C++

JUDIT A. BARKI

A MAJOR REPORT
IN
THE DEPARTMENT
OF
COMPUTER SCIENCE

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE
CONCORDIA UNIVERSITY
MONTRÉAL, QUÉBEC, CANADA

MAY 1995

ISBN 0-612-05140-4

CONCORDIA UNIVERSITY
School of Graduate Studies

This is to certify that the major technical report prepared

By: **Judit A. Barki**

Entitled: **Reusable Modules for Building Graphical User Interfaces under Motif and C++**

and submitted in partial fulfillment of the requirements for the degree of

Master of Computer Science

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining commitee:

____________________ Examiner

____

____

____________________ Supervisor

Approved ____________________
Chair of Department or Graduate Program Director

________ 19 ____ ____________________
Dean of Faculty

Abstract

Reusable Modules for Building Graphical User Interfaces under Motif and C++

Judit A. Barki

The design of Graphical User Interfaces using the Motif Widget set is very common. We identify two tracks in the design and development of GUI. In one track the developer uses the UIM/X software package, which is a GUI builder tool based on Motif. In using UIM/X the learning time is high but once learnt, new user interfaces can be developed rather quickly. In the second track, a developer who has learnt C++ and is knowledgeable about Motif does not have to learn anything new, but uses a library of modules to build a graphical user interface. This report contributes to the second track by developing a Reusable Module Library (RML) using C++ and Motif.

As suggested by Douglas A. Young in "Object-Oriented Programming with C++ and OSF/Motif" our RML modules are defined as C++ object classes. These classes are divided into two categories, *component* and *abstract* classes. The component classes give rise to visible objects on the screen, whereas abstract classes are used for management purposes. The RML developed in this work contains 25 component classes and 11 abstract classes.

As a way of demonstrating the usefulness of RML, an implementation project was undertaken. The project titled CAS (Course Advising System) is intended to provide advice to Computer Science undergraduate students in their selection of courses for registration purposes. CAS is designed following the Object Oriented methodology, in particular the Object Modeling Technique presented by J. Rumbaugh *et al.*. The CAS project is divided into two parts for the purposes of implementation[1] and this report is concerned with user interface part. The graphical user interface objects

[1] "Design and Implementation of an intelligent Course Advisor System" by Kim Duong is concerned with the functional implementation part.

for CAS are composed from array of push buttons, toggle buttons, text fields, scroll-able lists and customized dialog boxes. These high level objects (modules) form RML.

The ultimate goal of this report is to make RML classes to be easily understood and used (reused) by C++ programmers. Towards achieving this goal, in this report, each RML class is described in a systematic fashion containing, a short description in English, attributes, methods, inheritance and component graphs. The implementation of CAS user interface explains how the RML classes can be used by a C++ programmer.

Acknowledgments

I would like to thank my supervisor Dr. T. Radhakrishnan for his advice, guidance, helpful comments he has given me throughout my studies, especially during the preparation and writing of this work. I greatly value his infinite patience he has shown when I was discouraged. Dr. Radhakrishnan is not only a supervisor, he deeply cares about his students and gives special attention to them even after their graduation. Thank you Dr.Radhakrishnan.

Special thanks goes to Kim Duong whom I worked together on the analysis of the advising process and the Course Advising System.

I would like to thank all my friends here at Concordia University, Maria Janto, Joanna Sienkiewicz, Rani Vallurupalli, Alicja Celer for their supportive friendship.

Last and the most, I give my special thanks to my husband who gave me all imaginable help and support during this time. His infinite love is always my greatest source of energy.
Finally I think of my children, Tamas and George, who are fun, and who are giving so much happiness to me.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Registration for students in any program offered by Concordia takes places prior to the beginning of every academic term and involves lots of preparation, effort and time of not only students but also staff. The right advice received by the student may increase his/her academic performance and have direct effect on duration of his/her studies.

1.1 The Bachelor of Comp. Science Program

Concordia University has four Faculties :

- Art and Science
- Commerce and Administration
- Engineering and Computer Science
- Fine Arts

Faculty of Engineering and Computer Science has five departments:

- Civil Engineering
- Electrical and Computer Engineering
- Mechanical Engineering
- Computer Science
- Centre for Building Studies

At the bachelor's level, Faculty of Engineering and Computer Science offers the following programs[Uni91]:

- Bachelor of Engineering degrees in Building, Civil, Computer, Electrical, Industrial and Mechanical Engineering
- Bachelor of Computer Science degree

To be eligible for the degree of Bachelor of Computer Science, B.Comp. Sc. students must complete a program of at least 90 credits. This program consists of the courses of the Computer Science Core (those are mandatory for all students) and the courses specified for the student's Option. The Computer Science department offers the following five options within B.Comp. Sc.:

- Information Systems
- Software Systems
- Systems Architecture
- Theoretical Computer Science
- General Science

A student can be a full-time or part-time student. Full-time undergraduates in Computer Science have to take 24 credits or more for Fall and Winter terms (Regular Session) and 12 credits or more for the Summer Session. Part-time undergraduates in Computer Science have to take less than 24 credits for both Fall and Winter terms (Regular Session) and less than 12 credits for the Summer Session. Regular Session is divided into a Fall Term (September - December) and a Winter Term (January - April), each of 15 weeks' duration, including an examination period. Summer Session covers all courses offered between the beginning of May and the end of August.

For the average student, one credit represents, a minimum of 45 hours per term spread across the activities of lectures, assignments, laboratories, studio or practice periods, examinations, and personal work.

1.2 Registration and Student Advising

Selection of courses for a session or a term is done during a specified period prior to the start of classes. Regular registration for the Fall and Winter terms normally takes place in mid-August and mid-December respectively.

Each student must program a schedule for a Fall or Winter term and obtain his/her transcript (except those students registering for the first time) prior to the registration. The student's schedule is then approved by an advisor. The advisor manually checks the selected courses and the transcript to see:

- If the prerequisites are satisfied.
- If the schedules do not conflict.
- If the credit load is that can be handled by the student.

The performance of each student in each course is evaluated by the instructor(s) responsible for that course at the end of a term. The transcript records the student's performance of each course taken and the grade point average (GPA). Based on the transcript, the advisor measures the level of achievement of each student and advises the student which courses to take.

In 1994 fall, Concordia introduced telephone registration called CARL, Concordia Automated Response Line. It performs the registration of qualified students and provides information regarding student accounts and academic performance (grades) over the telephone. However this dial- up system does not give advising. A Course Advising System, CAS, is needed to assist students in selecting courses and to reduce the advising work for the staff as well.

1.3 The Structure of B.Comp. Science Program

As stated under Section 1.1, the degree of Bachelor of Computer Science requires students to complete satisfactorily a number of courses of the Computer Science Core and Option. Each student must complete 35 credits of Computer Science Core and at least 55 credits of his/her particular Option courses. All courses in Computer Science Core and certain courses in each Option are mandatory, the others are optional. The Computer Science Core (CSC) provides fundamental study of each major area of computer science (system architecture, programming methodology, databases, computer architecture, software engineering, operating systems). The Option specific courses are composed using six sets of courses (sublists as we are going to refer to them). These sublists are shown in Table 2. The number of credits, the student is required to obtain from each sublist is indicated for each Option.

1.3.1 Information System Option (ISO)

In this option, students learn more on "the application of computers in business, with a special emphasis on databases, software engineering, and management information systems"[Uni94]. The degree requirement for this option is summarized in the ISO row of Table 1.

1.3.2 System Architecture Option (SAO)

In this option, students understand more "on aspects of the design of digital circuits, and their integration into computer architectures"[Uni94]. The degree requirement for this option is summarized in the SAO row of Table 1.

1.3.3 Theoretical Computer Science Option (TCS)

In this option, students focus "on numerical analysis and symbolic computation"[Uni94]. The degree requirement for this option is summarized in the TCS row of Table 1.

1.3.4 General Science Option (GSO)

In this option, students are allowed "to define an area of specialty within the sciences"[Uni94]. The degree requirement for this option is summarized in the GSO row of Table 1.

1.3.5 Software Systems Option (SSO)

In this option, students concentrate on "the design and analysis of large- scale software systems"[Uni94]. The degree requirement for this option is summarized in the SSO row of Table 1.

	CSC	CSE	LRE	OCS	OBR	OSE	LLAB
ISO	(35)	(3)	(6)	(9)	(30)	L1 (6)	COMP292
SAO	(35)	(3)	(6)	COMP361	(27)	L2 (6) L3 (9)	(1)
TCS	(35)	(3)	(6)	COMP361	(18)	L4 (6) L5 (9) L6 (9)	(1)
GSO	(35)	(9)	(6)	COMP361	(18)	L7 (18)	(1)
SSO	(35)	(5)	(6)	(19)	(18)	L7 (6)	(1)

Table 1: Degree Requirements

Note:

1. Each row of Table 1. represents the Degree Requirement of one option.
2. The column titles indicate the sub-lists of Degree Requirement.
3. The number in parentheses, eg. (35), denotes number of credits required for each sub-list (see the column title).
4. See Table 2. and 3. for details of courses in each sub-list.

1.4 The Process of Advising

The present registration procedure at Concordia University can be abstracted to contain three major steps. First the Course schedule and Registration form are sent to the students who can plan on their own schedule (select the desirable courses). Second, the collection of selected courses is verified, modified and finally validated as the result of the Advisor-Student communication. The third step is the actual registration in the selected courses. Our project deals with the second stage of the registration procedure namely the development of a Course Advising System (CAS).

Advising is to help students to get the "appropriate courses". Appropriate courses are those he/she might need to accomplish toward achieving the specified degree. The Advisor is an expert, a professor in the department that offers the degree the student is working for. The student prepares the list of preferred courses prior to advising. However this is not mandatory. The Advising procedure is two folded. It consists of verifying the selected course list and suggesting additional courses. Usually an Advising session is the mixture of the two. However, if the student does not prepare the selected course list in advance, the Advising session will consist of the latter only; if the student lists proper number of acceptable (qualifying) courses it will consist of the former.

1.4.1 The preferred course list is verified with respect to the followings

- Each course in the list has to be in accordance with (can be included into) the Degree Requirement specified by the student's option.

- Pre and co-requisite of each course in the list has to be satisfied
- The day and time of any course in the list has to fit in the schedule formed by the rest of the preferred courses. That is, conflict of times of the selected courses can not occur.

For a given student, courses not satisfying the first two conditions are taken out immediately without any further consideration. Attention has to be paid to co-requisite. If course 'A' is co-requisite of course 'B', removing course 'A' has to result in removing course 'B'. (see section 1.1.4. for exceptional cases) The verification of the selected course with respect to the schedule is more complex. If there is a conflict between course 'A' and course 'B' the Advisor and the Student have three choices:

1. Take course 'A' (or course 'B') in another time if it is offered. This means the student takes the same course but a different section.
2. Replace course 'A' (or course 'B') with a different course (course 'C'). Course 'C' has to qualify also, and the above three conditions have to be met.
3. Drop one of the conflicting courses ('A' or 'B') and do not suggest any replacement.

The curriculum requirements of a course can include Lecture, Tutorial, Laboratory. It follows that the Course Schedule includes:

- Day and Time of Lecture, if there is Lecture
- Day and Time of Tutorial, if there is Tutorial
- Day and Time of Laboratory, if there is Laboratory

Consequently a course fits into the Schedule of selected courses if all of its components fit in the Schedule. (see section 1.1.4. for exceptional cases)

1.4.2 Suggesting a course to add to or discard from the selected list

The Advisor can suggest additional courses if the student wants or has to extend the list of selected courses. Advisor may use various rules of thumb and dialog with the student. The workload, strength and weakness of the student, GPA, area of interest, preparation for graduate studies, present or future job market can all be considered during an advising session. Courses to be chosen to fulfill the elective part of the Degree Requirement always draw the Advisor's attention. The Advisor can suggest to discard a selected course due to heavy workload to help students performance or suggesting another course fitting better into the student's profile or interest.

1.4.3 Accountability for Suggested Courses

The Advisor has to approve by putting his/her signature on the resulting list of suggested courses. Yet the student has the final responsibility.

1.4.4 Exceptions

Some exceptional cases are presented in this section, when the advising procedures differ from the standard procedures. This section will be referenced throughout our work.

Categories of Students

Visiting Student: Students of other universities wishing to take one or more courses in Concordia. These students have an official letter signed by the responsible person specifying the courses he/she can attend.

Independent Student: Student is not accepted and registered in any degree program. The list of courses that can be taken is restricted to contain those specified by the department.

Schedule

Generally, if the curriculum of a course includes more than one kind of class (LAB, TUT, LEC) the student is required to take the classes belonging to the same section. For example in the case of COMP291 the curriculum of the course contains Tutorial (TUT) and Laboratory (LAB). Students taking this course should attend Tutorial and Laboratory belonging to the same section. Therefore the Course schedule to fit in the schedule of selected courses:

COMP291/XX TUT M 18:50 - 19:40
COMP291/XX LAB T 20:00 - 22:00

This is the clean schedule matching. However, it is possible, when essential, to mix the sections among the different kinds of classes of a course (one section's lecture and a different section's tutorial), but this has to be clarified during the advising session.

This description and analysis of the Advising process is based on the observation of the Undergraduate Student Advising as done in Computer Science Department in Concordia University as of August 16, 1994.

1.5 The Project

An intelligent system is developed to assist in the present Student Course Advising procedure. It performs the verification of suggested courses and handles the regular cases of advising. The project (Course Advising System) provides a framework for future research (rule base, knowledge base system development). The Course Advising System will be referred to as CAS throughout this report.

1.5.1 Problem Statement

The system is to prepare the list of courses the Student would take in the next year. This "list of courses" is the output of the system and will be called "Suggested Courses" or "Suggested Course List" interchangeably. The Suggested Course List can only contain courses which are offered by Concordia University. The Suggested Course List would be created based on the following input (Figure 1):

- Student's Degree Requirement: CAS is to suggest courses that can be used toward the fulfillment of B.Comp. Science.
- Student's Record (Transcript)
- Pre and co-requisite of courses
- Requirements based on Student's Status: Full time Student has to take at least 12 and not more than 16 credits per term. Part time Student has to take at least 3 and not more than 6 credits per term.
- Student's Preferences: Student can influence the output by entering various preferences. (Preferred courses, time and location, maximum and minimum

number of courses or credits etc.). Elective courses can be chosen according to the preference of the Student; determined interactively; or CAS can derive them using some heuristic rules.

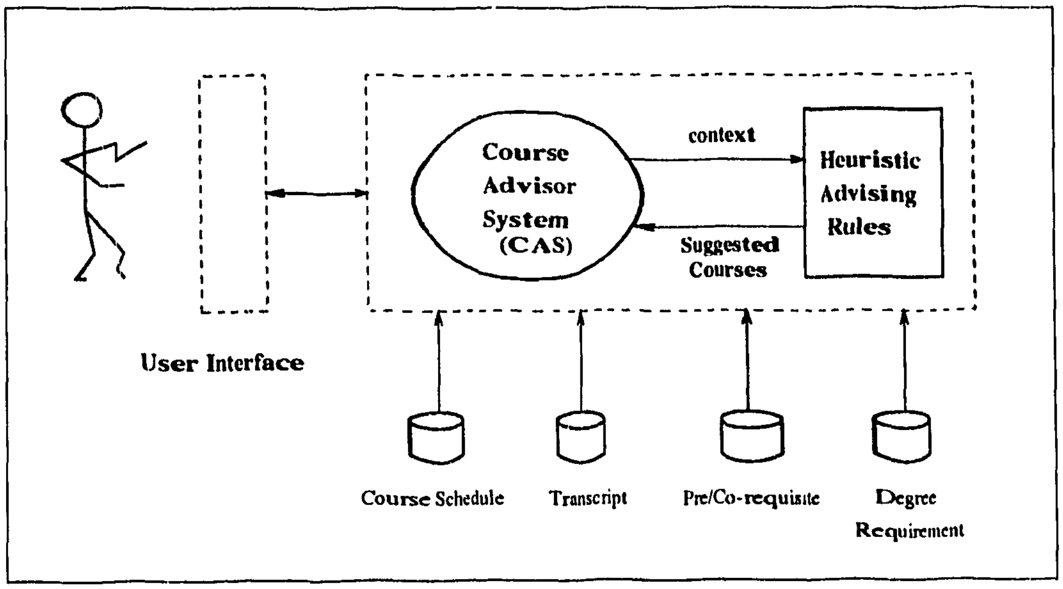


Figure 1: Overview of Course Advising System

Abbreviation	**Course List**	**Credits**
CSC - Computer Science Core courses for all options	{ COMP215, COMP220, COMP231, COMP245, COMP326, COMP335, COMP346, COMP352, COMP353, COMP354, ENCS281 }	35
CSE - Computer Science Elective courses for all options	{ COMPxxx or ENCSxxx , where xxx > 220 but including no more than 2 LLAB courses }	(ISO,SAO,TCS): 3; (GSO): 9; (SSO): 5;
LLAB - Language Lab course	{ COMP291, COMP292, COMP293, COMP294, COMP295, COMP296, COMP297, COMP298 }	1
LRE - Least Restrictive Elective courses for all options	Chosen from any department	6
OBR - Option Based Required courses	ISO = { ACCO 213, 218, ECON 201, 203, FINA 214, MANA 266, MARK 213, DESC 243, 244, 250 }	30
	SAO = { EMAT 212, 232, 252, 312, ENGR 273, 274, 371, ELEC 311 }	27
	TCS = { MATH 242, 243, 262, 263, 282, 283 }	18
	GSO = { MATH 242, 243, 262, 263, 282, 283 }	18
	SSO = { MATH 242, 243, 262, 263, 282, 283 }	18
OCS - Option Specific Computer Science courses	ISO = { COMP 445, 451, 457, 458, 472, 474 }	9
	SAO, TCS, GSO = {COMP 361 }	3
	SSO = { COMP361, 442, 445, 446, 451, 465, 485 }	19
OSE - Option Specific Elective courses	ISO = { L1 }	6/L1
	SAO = { L2, L3 }	6/L2, 9/L3
	TCS = { L4, L5, L6 }	6/L4, 9/L5, 9/L6
	GSO = { L7 }	18/L7
	SSO = { L7 }	6/L7

Table 2: Sub-list structure of Degree Requirements

Name	**Definition**
L1	Courses having prefixes ACCO, ECON, FINA, MANA or MARK except ACCO 220, 221, MANA 211
L2	{ COMP 327, 421, 445, ENCS 455 }
L3	{ ENCS 245, 456, ENGR 372, 471, ELEC 312, 341, 442, 461, 462 }
L4	{ COMP 441, 465, 467}

L5	Chosen from one of the following lists :
	list1: { MATH 322, 381, 392, 393, 394, 432, 491, 492 }
	list2: { MATH 231, 312, 336, 337, 381, 432, 435, 436, 437 }
	list3: { MATH 342, 343, 348, 351, 353, 448, 451, 454 }

L6	{ MATH 271, MATH [312..397] , MATH [425..499]}
L7	Courses chosen from : • Faculty of Arts • Faculty of Commerce and Administration • Faculty of Engineering and CS other than COMP201 • Other courses authorized by the office of Associate Dean

Table 3: Course Lists

1.5.2 Scope of the System

This system will perform the Undergraduate Advising at the Department of Computer Science of Concordia University.

There are five options in undergraduate Computer Science:

- Information System
- System Architecture
- Theoretical Computer Science
- General Science
- Software System

The Student's option, status and the courses already taken can be read from the current transcript of that student along with many other useful information. The databases supporting CAS should be updated before every registration period.

Chapter 2

The Object Modeling Technique

In our work, the object -oriented paradigm is used, especially the methodology that is promoted in [RBPL93] and referred to as Object Modeling Technique (OMT).

2.1 Overview of OMT

The dominant characteristic of OMT methodology is the application of "an object-oriented notation to describe classes and relationships throughout the life-cycle" [RBPL93]. As the very first stage of the software development process, an Object Model is created. Every subsequent stage contributes to this Object Model until implementation can be carried out with the desired amount of effort. That is, a more detailed Object Model means a faster, easier implementation. OMT takes into consideration the two other orthogonal views of a system with the creation of the Functional Model and Dynamic Model. However Object Model is viewed as the most important one and the implementation is based on its refinement, extension and optimization. There are three stages of the OMT methodology. Analysis, System Design and Object Design.

2.1.1 Analysis

According to the well known software engineering principle for designing a piece of quality software the first step should be the preparation of a Software Requirement Document (SRD). In OMT methodology the Analysis Document (produced during analysis and served as SRD) contains the followings:

Problem Statement: States the requirements.

Object Model: Defines the static data structure of the system by means of object classes, their attributes, operations and relationships to each other. The Object Model consists of:

- Data Dictionary: Lists the defined object classes with their short definition.
- Object Model Diagram: Shows the relationships between object classes. Attributes and inheritance are also shown.

Dynamic Model: Describes the time-dependent behavior of the system by showing how object classes respond to the stimulus from other objects. The Dynamic Model consists of:

- State Diagrams[1]: For each object class a state diagram is constructed to show dynamic behavior.
- Global Event Flow Diagram: Shows all events between the object classes. Events between class 'A' and class 'B' are listed on the arc connecting 'A' and 'B' regardless of any sequence of the occurrences.

Functional Model: Describes what happens to the objects established in the Object Model. The Functional Model consists of:

- Data Flow Diagrams: Determines the input, output and the functions that relate them.
- Constraints: List of constraints or invariant that the analyst should take into account in the design phase.

2.1.2 System Design

High level design decisions (accessing resources, identifying subsystems, management of data stores, tradeoffs, single or multiple threaded control etc.) are made during this phase. Namely the system architecture is designed and recorded in the System Design Document.

[1]state diagram: "Relates events and states." event: "An individual stimulus from one object to another." state: "Specifies the response of the object to input events." [RBPL93]

2.1.3 Object Design

The Object Model derived during Analysis serves as "skeleton" for the Object Design phase. During this phase all three models are refined. The output is the Design Document containing detailed versions of the three models. However the primarily purpose of Dynamic and Functional model refinement is to define additional operations or attributes for the object classes (introduce new ones) to facilitate implementation and carry out the necessary computation and control. Hence a sufficient Design Document consists of:

- Detailed Object Model:
  - All attributes and methods (operations) are identified.
  - Data structures are decided.
  - Operations (practically functions) are described by pseudo code or other means along with their arguments.
  - Applicable constraints and explanations are recorded in a reserved section in each object class.
- Transfer function (main function) description by pseudo-code or other means.
- Uses Diagram: Shows the interaction of the objects or subsystems.

2.2 Analysis of CAS

2.2.1 Problem Statement of CAS

See sections 1.5.1 and 1.5.2

2.2.2 Object Model of CAS

Based on the Problem Statement (Section 2.2.1), a set of object classes is defined through the following stages:

1. Identifying object classes by extracting nouns from Problem Statement of CAS.
2. Identifying attributes of object classes and associations between object classes.
3. Simplifying object classes using aggregation and inheritance.

4. Identifying main operations of each class.

The outcome of the above steps fina ly produces three pieces of documents, namely Class Description Dictionary, Definition of Object Classes and the Object Model.

Class Description Dictionary of CAS

This document contains the names of all object classes, presented in alphabetical order, each of which is followed by a short description.

CAS Course Advisor System suggests a list of courses to be taken during a predefined period. CAS communicates with the student and consults with the student's transcript. It prepares the course list with respect to Option, Status of the student, courses already taken (Transcript), degree Requirement, Schedule.

Course can be taken by the student. All relevant information regarding a course is recorded (course, Credit, Pre/CO-requisite).

Course list is a set of courses. A Course list contains zero, one or many courses. Course list class is an aggregate of object class "Course".

Course schedule The schedule of the courses that have identical "Course ID". The class Course schedule is an aggregate of object class "Schedule Entry"

DegreeRequirement List of all courses must be completed in order to satisfy the requirement of a specific degree. Depending on the student's option the requirement consists of one list from each of the following course lists: CSC, CSE, LAB, OSE, OCS, OBR, LRE

Explanation Messages to explain to the student the reason for not suggesting his/her preferred courses. The class Explanation is an aggregate of object class "Explanation Entry".

Explanation entry It is explanation provided regarding one particular course. It consists of several flags to indicate the type of rejection.

Schedule entry Schedule of one particular course, which can be identified by (Course ID, Section).

Student Student is registered to obtain a degree in a scientific field belonging to the scope of the present project. Student is the user of the Course Adviser System.

Suggested Courses class is the output of course adviser system. It is a suggestion for the student what courses he or she should take next. This consists of the list of courses to be taken by the student together with their schedule entries.

Term transcript records the courses that were taken during a term. Term transcript consists of the identification of the term (Term/Year) and an ordered collection of "Transcript Entry". The "Transcript entries" are arranged in ascending order on the "Course ID" (prefix, course No.). Term transcript class is an aggregate of object class "Transcript entry".

Transcript is an officially maintained report of the academic activities of the student. The transcript contains an ordered set of paragraphs ("Term Transcripts"), and information about the program by the means of class attributes. Transcript class is the aggregate of object class "Term transcript".

Transcript entry is created for every course that the student registered for and did not withdraw before the academic withdrawal deadline. A "Transcript entry" holds the following information: Course ID, Term, Section, Credit, Grade, Credit granted. Transcript entry class is an aggregate of object class "Course" with the constraint that pre/co-requisite is suppressed.

UserConstraint is specified by the student prior to the course suggesting process.

Table 4: Class Description Dictionary of CAS

Definition of Object Classes of CAS

This document contains for each object class the name together with its attributes and operations.

Class Name: CAS

Attributes:

- Current ID#
- Current Transcript
- Current Degree Requirement
- List of constraints
- List of suggested courses together with course schedule

Operations:

- Initialize CAS before advising
- Clean up CAS on exit
- Get all courses allowed to take
- Filter allowed courses toward the constraints specified by the student
- Perform suggesting courses
- Match course schedule
- Print suggested courses, explanation on the preferred courses

Class Name: Course

Attributes:

• Course	
• Prefix	COMP
• Course No.	224
• Credit granted	3
• Requisite	COMP244; COMP285
• Title	Introduction to Systems Programming

- Course Description   Basic machine organization ...

Operations:

- GetCredit
- GetRequisite
- GetTitle
- GetDescription

Class Name: Course list

Attributes:

- Number of courses in the list

Operations:

- All basic functions to provide list operations
- Load course list from course database
- Unload course list when done

Class Name: Course schedule

Attributes:

- Number of schedules in the list
- Course ID (prefix + Course No.)

Operations:

- All basic functions to provide list operations
- Load schedule list from course schedule database
- Unload schedule list when done

Class Name: Degree Requirement

Attributes:

- Effective period for the degree requirement (from starting year to ending year)

- Option
- List of CSC courses and the number of credits required from the list
- List of CSE courses and the number of credits required from the list
- List of LRE courses and the number of credits required from the list
- List of OSC courses and the number of credits required from the list
- List of OBR courses and the number of credits required from the list
- List of OSE courses and the number of credits required from the list
- List of LAB courses and the number of credits required from the list

Operations:

- Functions that retrieve the attributes

Class Name: Explanation

Attributes: None

Operations:

- Show the explanation to the student
- Update the data

Class Name: Explanation entry

Attributes:

- Course No.
- Prefix
- Section
- Suggested course is suggested (yes or no)
- Requisite requisite is Ok or Not or N/A
- Available yes or no
- Conflict no or course ID of conflicting course
- ForDegree yes or no

Operations:

- Get attributes
- Write attributes

Class Name: Schedule entry

Attributes:

• Term	2
• Class Name	Lec
• Section	AA
• Day	M-W—
• Starting time	18:30
• Ending time	21:45
• Location	H535-2
• Capacity	20
• Instructor	N/A

Operations: Functions that retrieve the schedule line info

Class Name: Student

Attributes:

- Name
- ID#
- Phone#
- Option
- Date of First Registration

Operations: Functions that retrieve the attributes

Class Name: Suggested Courses

Attributes:

- Course ID (prefix + course no.)

- Course instructor
- Campus
- Location
- Class
- Day
- Time
- Suggested flag to indicate whether the course is advised to take.
- Explanation text if the course is advised not to take.

Operations: Functions that retrieve the suggested courses info.

Class Name: Term transcript

Attributes:

- Term, Year
- List of transcript entry of taken courses in the specified term.

Operations:

- Calculate GPA for the specified term
- Count number of credits of taken courses in the specified term
- Get list of all failure courses

Class Name: Transcript

Attributes:

- First registration date
- Minimum credit to take
- Status (full-time or part-time)
- Option
- Time limit, to finish the degree
- Program status
- Last GPA

- Commulative GPA

Operations:

- Functions that retrieve the transcript line information.
- Read transcript

Class Name: Transcript entry

Attributes:

- Course ID (prefix, course no.)
- Section
- Credit granted

Operations: Functions that retrieve the transcript entry information.

Class Name: User Constraint

Attributes:

- List of courses the student prefers to take in this term.
- List of courses the student does not want to take in this term.
- List of schedule day and time the student cannot attend.
- List of campuses the student does not prefer.
- Maximum and minimum number of courses the student wants to take.
- Maximum and minimum number of credit the student wants to take.

Operations: Read and update the user constraint specified by the student

Object Model Diagram of CAS

As mentioned in 2.1.1, the attributes are usually shown along with the object classes in the Object Model Diagram. To increase readability of the diagram, only the qualifiers[2] are indicated. All attributes and operations can be looked up in the Definition of Object Classes document. Hence the Object Model Diagram of CAS (Figure 2.)

[2]A qualifier is an attribute to distinguish among a set of objects at the many end of a many-to-many, many-to-one or one-to-many association.

shows relationships, aggregation and inheritance (see [RBPL93] for notation). The following information can be read from the diagram:
CAS object accesses:

- a selection of "Course" objects via their "Course #" and "Prefix" attributes
- a selection of "Course Schedule" objects via their "Course#" and "Prefix" attributes
- a selection of "Transcript" objects via their "Student ID#" attribute
- a selection of "Degree Requirement" objects via their "Option" and "DateOfFirstRegist" attributes

CAS object produces:

- a "Suggested Courses" object

Student communicates with:

- CAS object

Student enters:

- User Constraints

Student has:

- "Transcript" object

Student receives:

- "Suggested Courses" object
- "Explanation" object

"Suggested Courses" object class inherits all attributes and operations from "Course List" object class.
"Degree Requirement" object class is aggregated from 7 object classes. These 7 classes inherit all attributes and operations from "Course List" object class. Explanation of these course lists can be found in Chapter 1., Table 1. and Table 3.
"Course Schedule" object class is an aggregate of many "Schedule Entry" object classes.

"Course List" object class is an aggregate of many "Course" objects.

"Explanation" object class is an aggregate of many "Explanation Entry" objects.

One "Course" object is aggregated into a "Transcript Entry" object.

"Term Transcript" object class is an aggregate of many "Transcript Entry" objects.

"Transcript" object class is an aggregate of many "Term Transcript" objects.

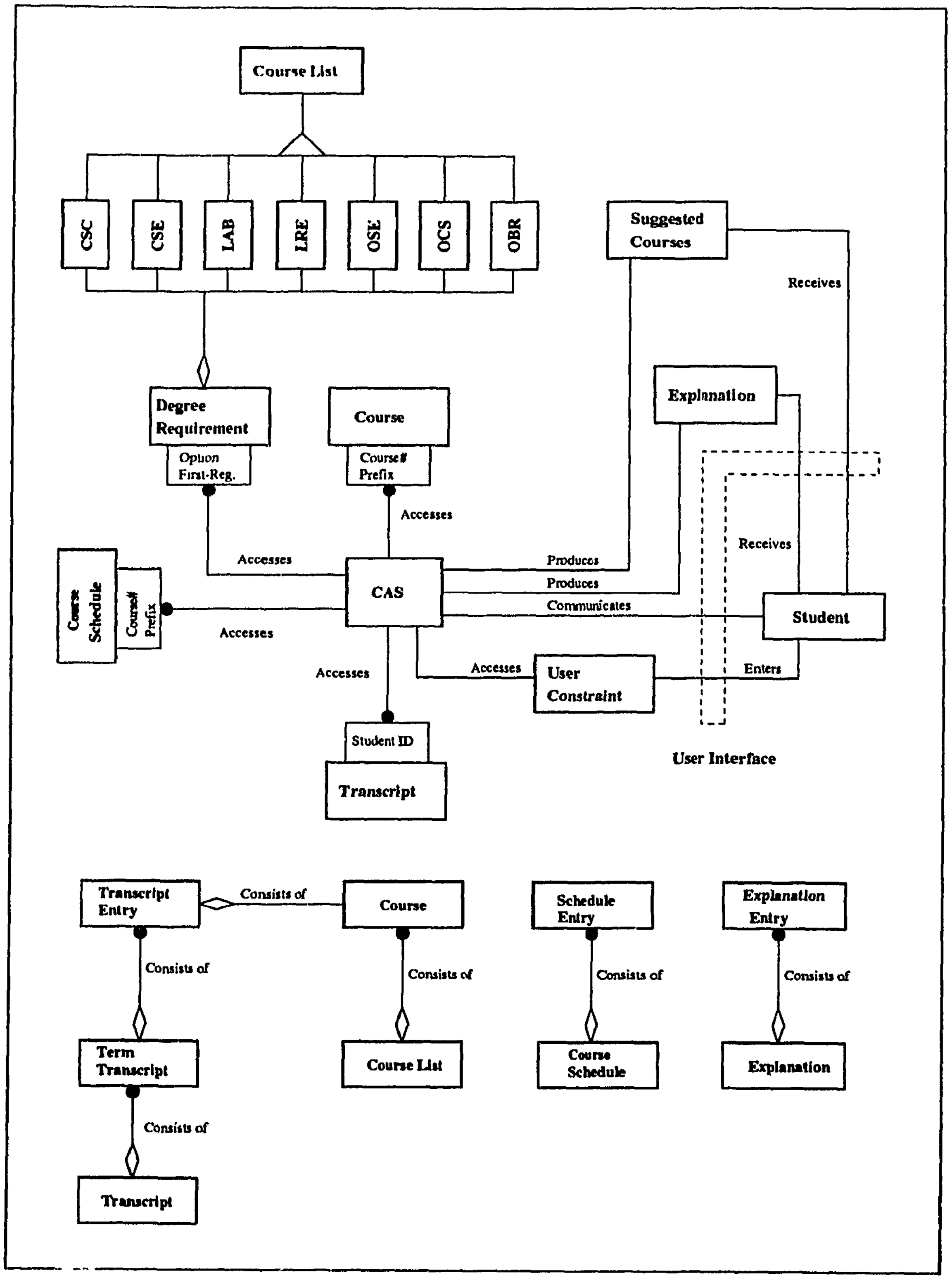


Figure 2: Object Model Diagram of CAS

2.2.3 Dynamic Model of CAS

The goal of this stage of analysis is to determine the State Diagrams of the objects and the Global Event Flow Diagram.

Scenarios

The process to produce the two pieces of document is set out preparing scenarios[3] to cover all typical sequences of user-system interaction.The importance of scenarios lies behind its understandability by users (clients) and developers as well. For this reason we suggest to include the collection of scenarios used to develop a software into the Dynamic Model.There has to be a scenario for each of the normal and exceptional cases of operation. Exceptional cases are when not expected, out of range or erroneous user input occurs, all remaining cases are considered as "normal operation". It is assumed that the User Interface of CAS Figure 1. screens the user input for acceptable format and range (except for valid student ID#) The scenarios describing CAS can differ in the following actions:

i. Student enters preferred course(s):

- yes
- no

ii. Student enters some constraints:

- yes
- no

iii. Student's confirmation of suggested courses:

- accepts entire list
- rejects entire list
- modifies list

iv. CAS accepts the modification:

- yes

[3]A **scenario** is a sequence of events occuring during a meaningful information exchange between system and outside agent.[RBPL93]

- no

v. Student wants additional courses after point iv.:

- yes
- no

Entering preferred courses and constraints can be drawn together since preferred courses are special constraints. In the scenarios it is written "The Student enters the courses he/she prefers to take. The Student enters his/her preferences and constraints regarding: campus, workload , time, courses that he/she does not want to take.". Naturally students can enter constraints without entering preferred courses and vice versa. However the information is evaluated by CAS before proceeding so it is unnecessary to establish separate scenarios for i. and ii. From the permutation of ii. and iii. 6 scenarios are created. Out of this 6 scenarios, 2 scenarios are involved with iv. expanding the total number of scenarios to 8. Each of the 8 scenarios is involved with v. expanding the total number of scenarios to 16. There are two additional things to consider. First an abort operation that would let the user abort the advising session will let CAS return to its initial state: "The CAS asks the Student to type in his/her student ID#." Second a cancel operation that would let the user stop the present iteration of advising. Actually it would clear the already suggested courses, but retain the information previously input by the student such as ID# and data extracted from transcript, preferred courses and user's constraints. These two operations can occur at anytime during execution of any scenario, with the same effect. 10 scenarios that form a subset of the 18 scenarios that have been set up to describe CAS can be seen in Appendix A.

Event trace diagrams

Each scenario can be shown in an event trace[4] diagram. The event trace diagram is a collection of vertical lines corresponding to objects, and horizontal arrows indicating the events occuring between objects.Event trace diagrams can be generated by identifying events between objects in the scenario and representing them in the sequence of the occurrences [RBPL93]. Figure 3. shows the event trace diagram of Scenario 8. [Appendix A] as an example. It can be seen that the events occuring between two

[4]**event trace** is "an augmented scenario", that includes "an ordered list of events between different objects".[RBPL93]

objects, for example Student and CAS, while the system follows Scenario 8. are all assigned to the column bounded by the vertical lines representing the two objects, Student and CAS:

> Request ID#(from CAS), Enter ID#(from Student),
> Start advising(from Student), Display suggested courses(from CAS),
> Reject suggested courses(from Student), Start advising(from Student),
> Display suggested courses(from CAS),
> Confirm suggested courses(from Student),
> Print suggested courses(from CAS), Request ID#(from CAS).

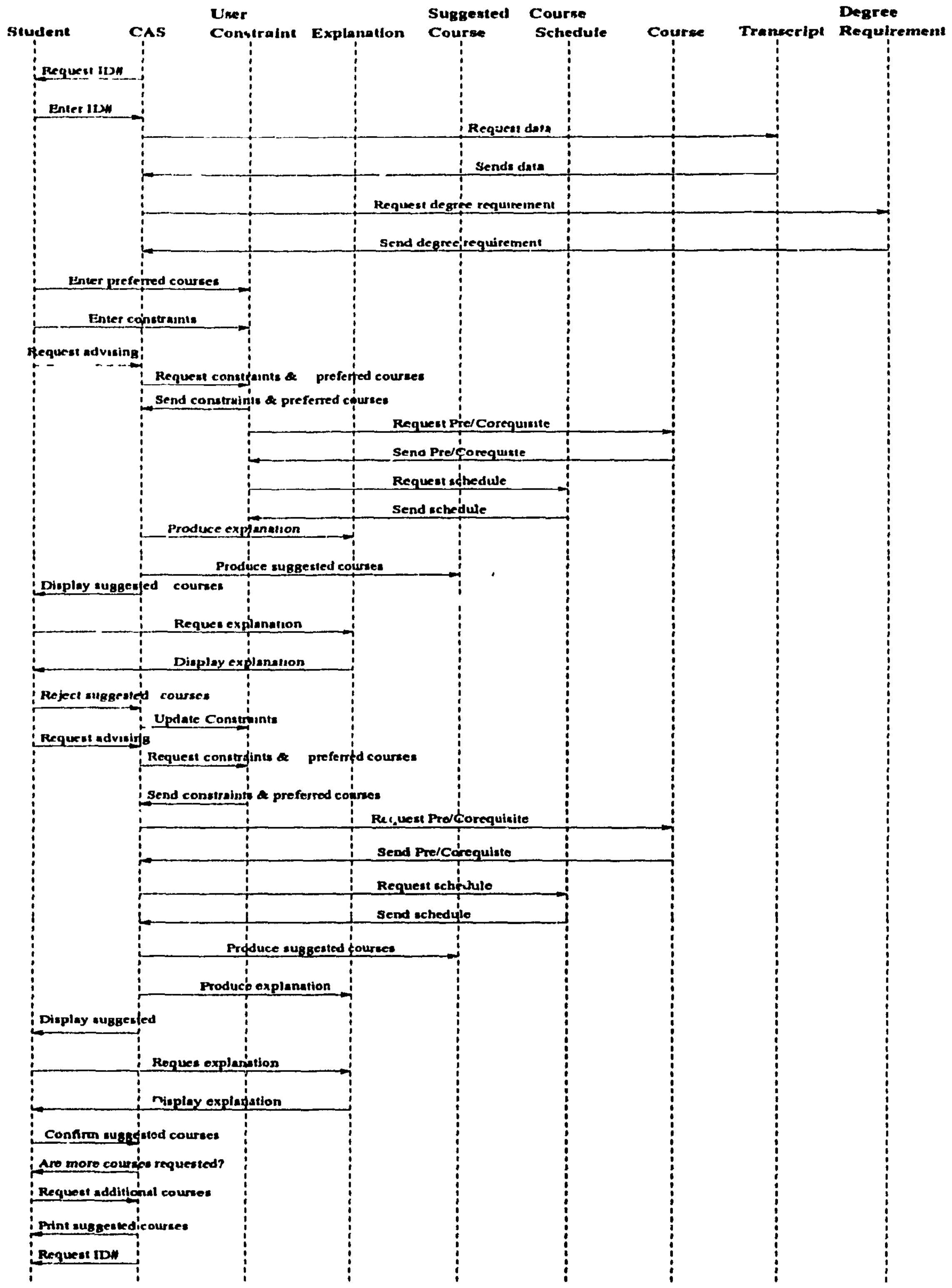
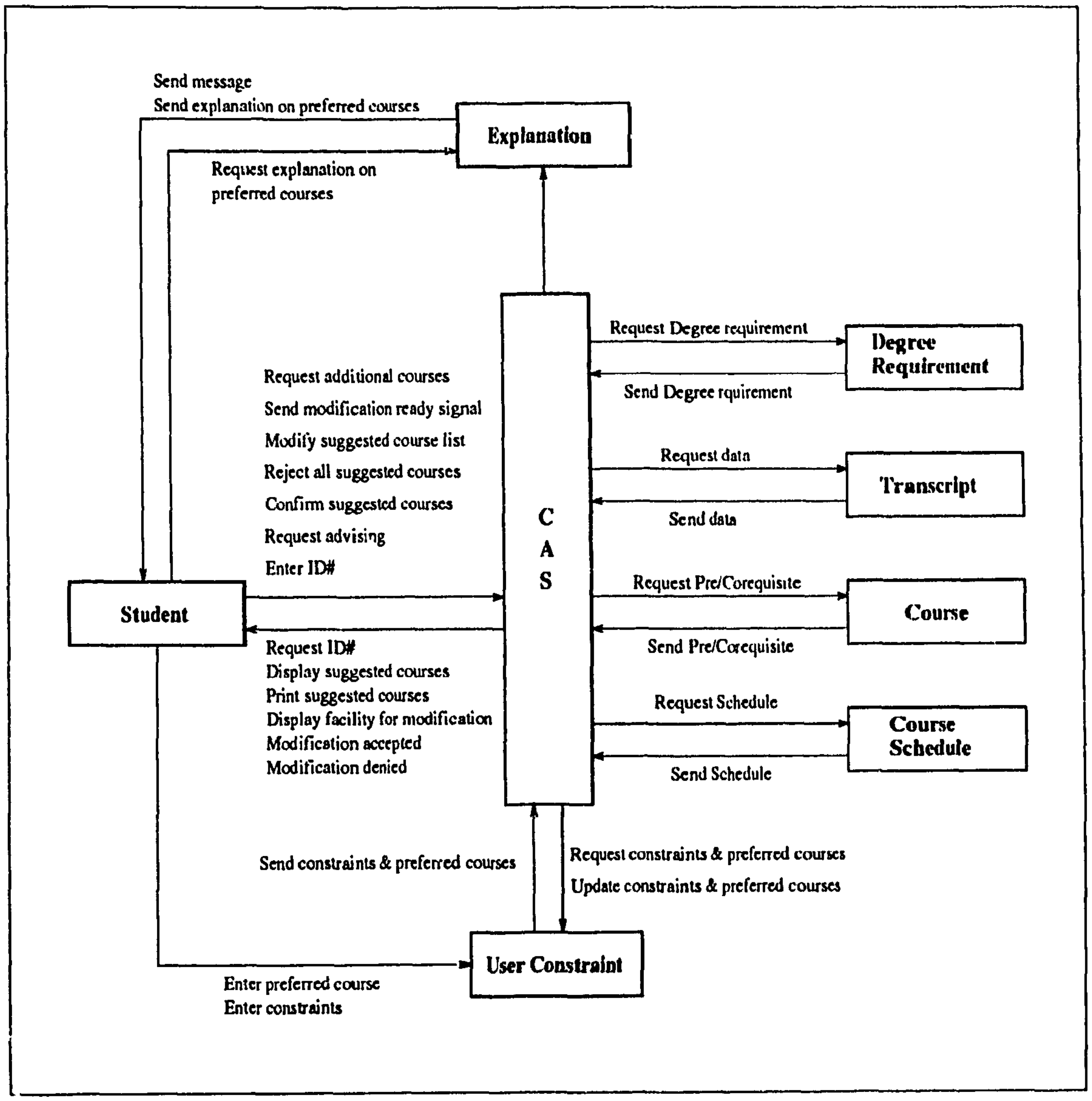


Figure 3: Event trace diagram based on scenario 8 [Appendix A]

Global Event Flow Diagram of CAS

Event Trace Diagrams are created for each Scenario. Then the events occuring between specific objects are taken from the appropriate column of each event trace and listed on the links of the event flow diagram of CAS (Figure 4.).



Send message
Send explanation on preferred courses
Explanation
Request explanation on preferred courses
Request Degree requirement
Degree Requirement
Send Degree rquirement
Request additional courses
Send modification ready signal
Modify suggested course list
Reject all suggested courses
Confirm suggested courses
Request advising
Enter ID#
C A S
Request data
Transcript
Send data
Request Pre/Corequisite
Course
Send Pre/Corequisite
Student
Request ID#
Display suggested courses
Print suggested courses
Display facility for modification
Modification accepted
Modification denied
Request Schedule
Course Schedule
Send Schedule
Send constraints & preferred courses
Request constraints & preferred courses
Update constraints & preferred courses
User Constraint
Enter preferred course
Enter constraints

Figure 4: Event flow diagram of CAS

State Diagrams of CAS

State diagrams have been created for "each object class with nontrivial dynamic behavior"[RBPL93] from the scenarios and event traces. All the event traces are merged into the state diagram belonging to each class. "CAS" object class has the most complex state diagram (Figure 6). Figure 5. shows the refinement of **Suggesting** and **Write Constraint** states of CAS object class. "Degree Requirement", "Transcript", "Course Schedule", "Course" object classes have mainly static characteristics. They serve as data storage, their operations involve mostly data access(request data, send data) and have one state. Upon receiving "request data" message, the appropriate data is selected and sent to the requester. "Explanation" and "User Constraints" object classes have two states depending on the input which can be "request data" or "write data". See Figure 7.

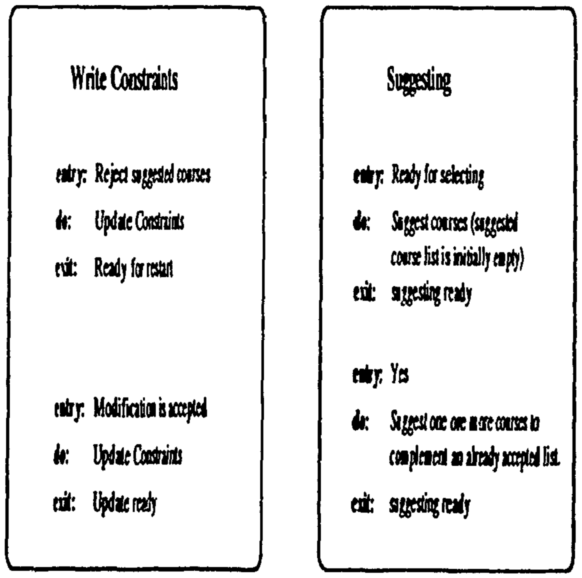

Write Constraints
entry: Reject suggested courses
do: Update Constraints
exit: Ready for restart
entry: Modification is accepted
do: Update Constraints
exit: Update ready
Suggesting
entry: Ready for selecting
do: Suggest courses (suggested course list is initially empty)
exit: suggesting ready
entry: Yes
do: Suggest one one more courses to complement an already accepted list.
exit: suggesting ready

Figure 5: State refinement of "Write Constraint" state of CAS object class

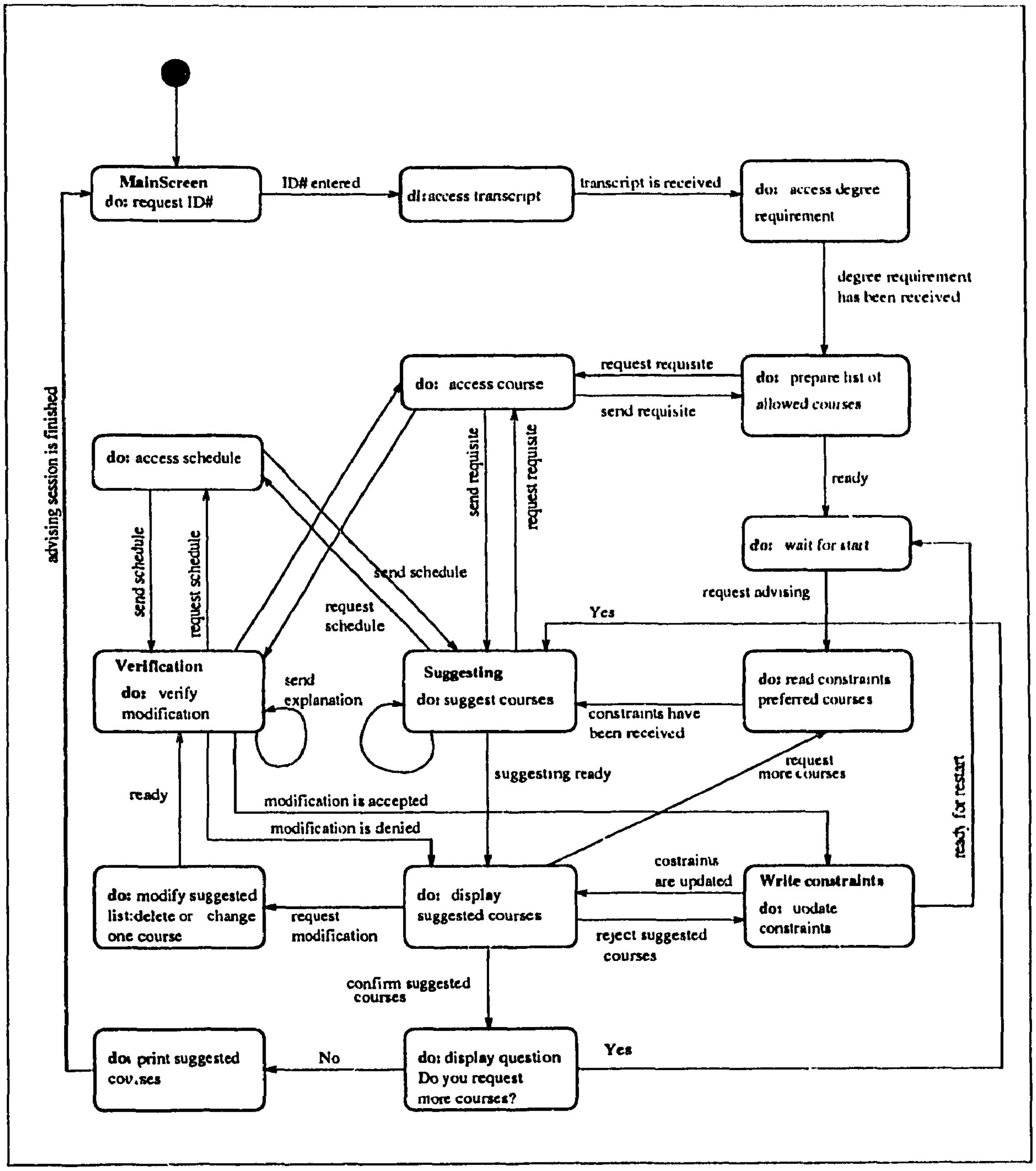


Figure 6: State diagram of CAS object class

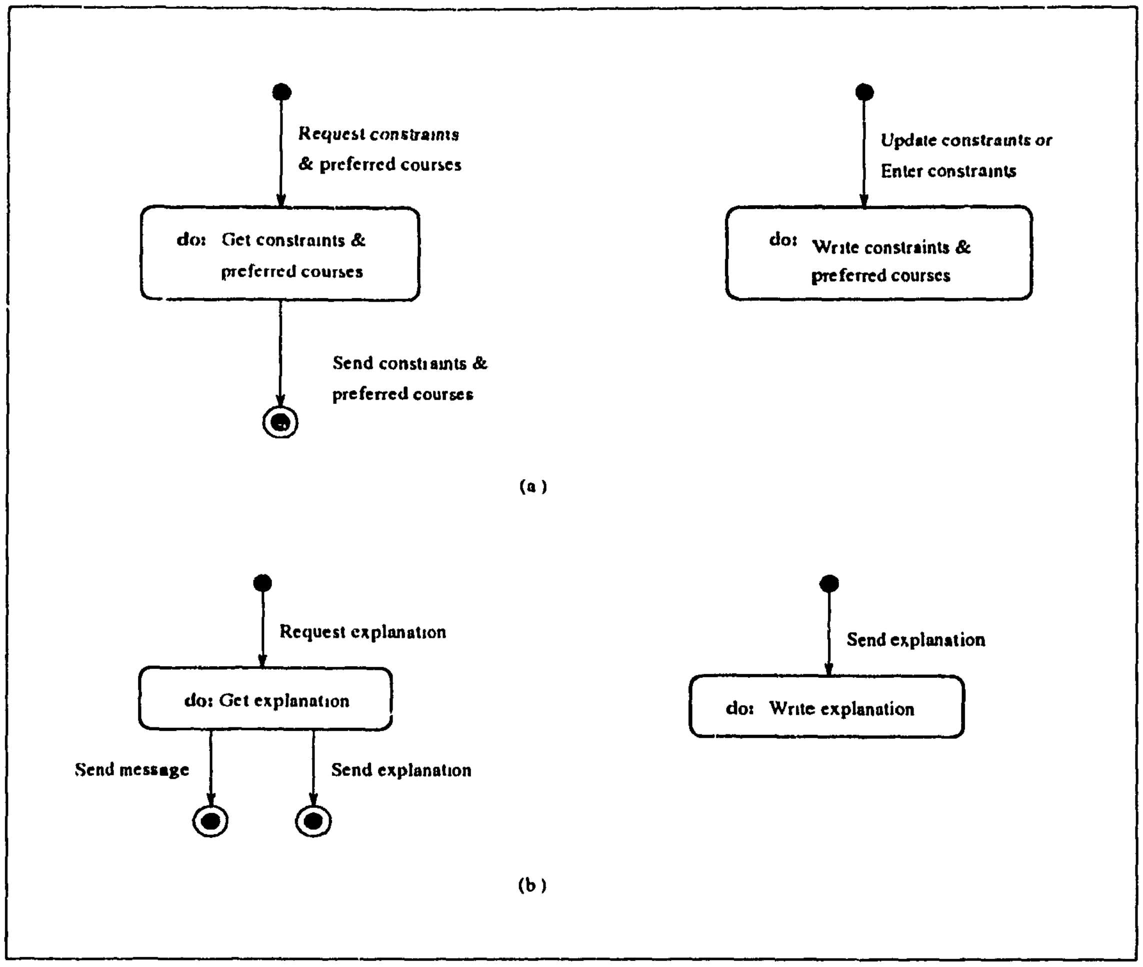


Figure 7: State diagrams of (a.) User Constraint, (b.) Explanation object classes

Functional Model of CAS

The student has to enter three things: Student ID#, Constraints and Preferred courses, Confirmation of suggested courses. These are the input to the system. The student will receive two things: Suggested courses with their schedule and Explanation on Preferred courses. These are the output from the system. Both agents receive messages from each other. The input and output of the system is shown in Figure 8.

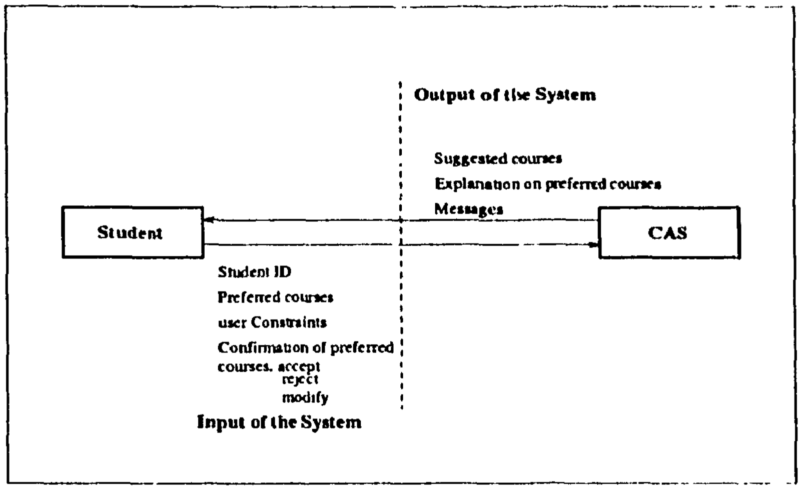


Figure 8: Input and Output values for the Course Advisor

The top-level data flow diagram (Figure 9.) shows the permanent data bases (Transcript, Degree Requirement, Course, Course Schedule) the system has to access, and the objects the output is stored with the data flow. Grades, Status, Option, the list of Taken courses and the date of 1st registration are read in from Transcript object and passed to "Perform course suggestion" process. This process retrieves the Degree Requirement object corresponding to the Student's Option and date of 1st registration. It obtains information regarding Pre/Co-requisites of certain courses from the Course objects and retrieves the Course Schedule objects corresponding to specific Courses. "Perform course suggestion" process obtains additional information that has been entered by the Student (Preferred courses, Unpreferred courses, Inconvenient time, Maximum and minimum number of credits and courses) and stored in User Constraint object. The output of the process is a list of courses and their schedules that are written into Suggested Courses object. Additional to this, explanation is prepared (and recorded in the Explanation object) to show the Student that his/her Preferred courses have/have not been selected. The Suggested courses and Explanation are displayed to the student.

The refinement of "Perform Course Suggestion" process is shown in Figure 10. The process is broken down into four processes to carry out the necessary computations:

Select Courses not yet taken: Takes the Student's Degree Requirement and the courses he/she has already taken and produces a list of courses that the Student still has to take (Required courses list).

Select Courses allowed to take: For each course in Required courses list, this process checks if the Pre/Co-requisites are satisfied and produces Allowed courses list.

Apply user constraint to allowed courses: Filters Allowed courses list with "Unpreferred courses" that is an attribute of User Constraint object. The resulting list is the Filtered allowed courses.

Selection, Match schedule: Selects courses from Filtered allowed courses list with respect to the remaining attributes of User Constraint. The schedules of selected courses has to be matched. The number of selected courses depends on the Student's Status or the relevant attributes of User Constraint object.

The further refinement of the three models will be carried out during Design phase [Duo].

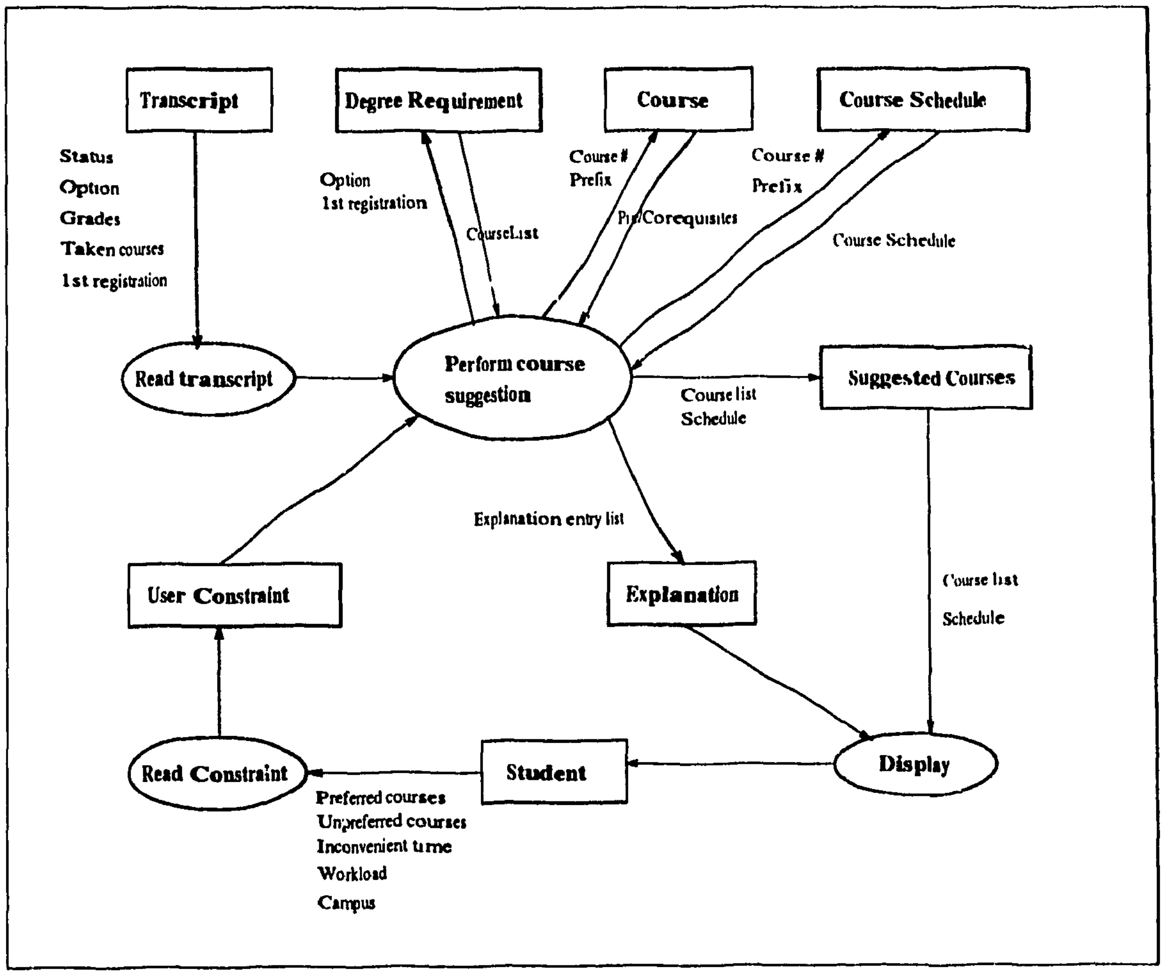


Figure 9: Top level Data Flow Diagram of Course Advisor

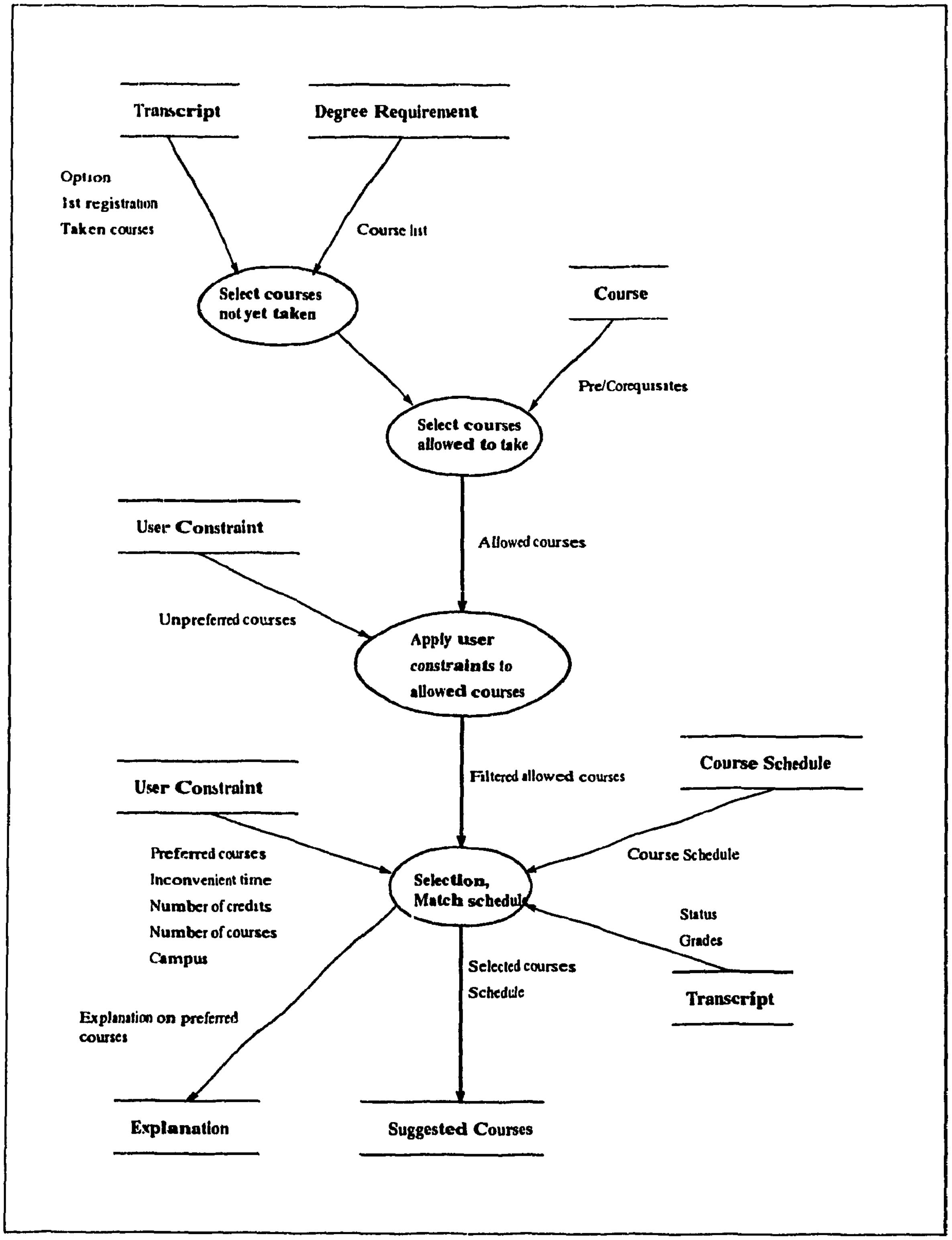


Figure 10: Data Flow Diagram for **Perform course suggestion** process

Chapter 3

Software Support for User Interface Development

The Course Advisor System is to be accessible via a Graphical User Interface. We implement this interface using Reusable Module Librai,/ that was developed as the part of this major technical report to support application developments based on C++ and Motif toolkit.

3.1 X Window System and OSF/Motif

X Window System (known as "X") is "an industry-standard window system that provides a portable base for applications with graphical user interfaces" [You92]. Motif (more formally OSF/Motif) is a toolkit developed by the Open Software Foundation and specifies the common user interface features needed by applications based on X. The Motif applications are built on three distinct layers [HF94]:

- **Xlib** provides a low-level interface to the basic services of the underlying window system.[SG92]

- **Xt Intrinsics** is a higher level library to facilitate the use of Xlib by the programmers. Xt provides hooks for specific toolkits such as Motif, Athena, OpenLook to take advantage of Xt facilities. It defines some user interface component classes, (such as the different "Shell" classes), abstract base classes (Core classes) and a protocol to carry out the interaction between application and Xt components. Xlib and Xt are implemented in "C". "The user interface components supported by Xt are called widgets"[You92] .

- **Motif Widget Set** consists of a set of components for creating user interfaces. Manager widgets that handle screen layout (BulletinBoard, Frame, PanedWindow, RowColumn, ScrolledWindow etc.) and widgets to support user interaction (different kinds of buttons, ScrollBar, Text etc.). See [Bra92] for complete set of Motif widgets.

3.2 Motif based GUI development

Motif is a widely-accepted toolkit and it is easy to use. It is better and easier to develop a GUI using Motif rather than with Xt and Xlib. In this context UIM/X type of interface generation plays a still better role than Motif. UIM/X is an industrial software that requires a good amount of learning time. One of the objectives of UIM/X is to make the UI development using Motif widgets an easier job. In this report we use another approach and develop a set of C++ classes, 25 component classes and 11 abstract classes to be precise, named **Reusable Module Library** (RML). The Figure 11. shows how RML is placed. Each RML class makes use of several Motif widgets. RML is specialized for the development of GUI using customized dialogs, lists, menus and multiple buttons. A C++ programmer can include the RML as a part of his/her library and quickly build a User Interface as explained in this report.

3.2.1 UIM/X Graphical User Interface Builder

UIM/X is an excellent software package, a "second-generation GUI Builder" [Ltd93] that facilitates the development process mentioned above. The steps of GUI development using UIM/X are:

- Specify all components of the graphical interface: interactively create the components of the interface, set resources etc.

- Write callback functions (callback function is that function invoked in response to an event)

- Generate code: C or C++ (UIM/X allows one to access and modify the generated code)

UIM/X dramatically reduces the effort of programming, and the interactive creation of the interface directly supports rapid prototyping. However, the developer should spend considerable time for learning both Motif and UIM/X since good knowledge of Motif is essential when using UIM/X for a complex application.

3.2.2 C++ and OSF/Motif

With Object Oriented Design (OOD) becoming more and more popular in the industry, developers are going to use a language like C++[1] to implement their applications. We should observe in what way the OO features of C++ might assist the UI development process. Douglas A. Young in [You92] recommends the use of OO facilities of C++ to create "high-level user interface components". Our aim is to implement the key components that will support an application development using C++ classes and Motif widgets.

Analyzing the application domain, a set of classes can be developed and used as a specialized class library. This set of core classes should efficiently support rapid prototyping and fitting the GUI into the specific application domain. The main criteria in this selection of components is re-usability. An object class is defined by its attributes and methods so re-usability depends on how well these two are chosen. For instance, object class "A" with three pushbuttons Pb1, Pb2, Pb3 as attributes is not as much reusable as object class "B" with arbitrarily many number of pushbuttons. Furthermore, let us suppose that the use of "A" requires the user to specify for each button if it is to be displayed or not; whereas in class "B" pushbuttons are created by the "constructor" as needed. The use of "B" will reduce the amount of coding during certain types of application development.

Implementation issues:

In [You92] some implementation issues are mentioned when using C++ with Motif. The first is that, C++ functions should have a prototype to declare the types of their arguments. X and Motif functions do not have prototype since they were written in pre-ANSI C. Second, X and Motif functions should be declared as external C functions with "extern C" declaration. These issues have been taken care of in the Motif header files as of Version 1.1.

There is another kind of issue to mention. From the OOD point of view, it is

[1]Programming language derived from "C" providing direct support for object oriented programming.

important to incorporate callback functions into the object class as methods (member functions). In C++ there is a pointer named "this" that is accessible for self-referencing an object. Programmers seldom need to use it explicitly, however "this" is passed as a hidden parameter to member functions (except static member functions[2]) every time they are called. Motif callback functions must have three parameters of predefined type as explained in [Bra92], so the hidden "this" makes the use of member functions as callback function problematic. A static member function does not have the hidden "this" parameter so [You92] suggests to use them as callbacks and call the appropriate member functions from it. "this" should be passed as an argument to the static function to make referencing the object possible.

3.3 Development of Reusable Module Library

Based on what is described in Section 3.2.2, we have developed a Reusable Module Library (RML). See Figure 11. RML consists of Abstract classes and Component classes. Component classes are built with a collection of Motif widgets. We studied the user-system interactions in CAS and designed component classes that could be used to build interfaces for similar applications. More precisely, the events of user-system interaction were examined and extracted from the Dynamic Model of CAS (See Section 2.2.3). These events can be classified as:

- application needs to display lists of data, structured in a table
- application needs to display messages to users
- user needs to enter lists of data
- user needs to select from predefined sets of data
- user needs to enter predefined commands to the system

The most important feature is a list of data, which can be displayed in a ScrolledList[3]. Augmenting the ScrolledList with a table header yields the object class of type Table. Adding a TextField widget and a set of buttons supports data manipulation on the ScrolledList (user can enter, delete, select items). The user can enter commands

[2]"only one copy of the function is stored for all instantiations of a given class" [Str94]

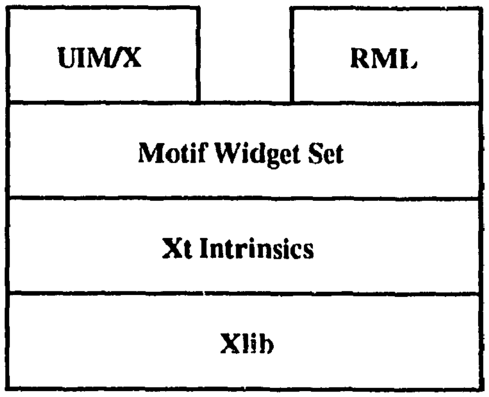
[3]ScrolledList is a compound object that we obtain "by creating a List widget as the child of a ScrolledWindow" [HF94]

to the system through menu, pushbuttons and toggle buttons. The system displays messages to the user with Message Dialogs or in a Text widget. Customized Dialog boxes facilitate the information display.

Abstract classes are "to define a common interface for its subclasses" and can not be instantiated according to [EHJV95]. We relaxed this definition in the case of RML, so some abstract classes can be instantiated (this will be indicated at the description of the object classes in Section 3.4). Additionally we can say that Abstract classes of RML are not visible on the screen, whereas its Component classes can always be visible.

Abstract classes of RML serve the following purposes:

- Support the various functionalities of Component classes such as external call-back functions.
- Collect the reusable aspects of Component classes through inheritance.
- Perform object management functions.



A layer can access the services of any layers below it.

Figure 11: Library Architecture of UIM/X and RML

Several Component and Abstract object classes have been developed based on the above discussion. They are listed below for a quick overview and described in detail in Section 3.4.

Component object classes

1. ActiveEditTable	Table and a collection of lists and pushbuttons
2. ActiveList	List, one text-field and a set of toggle buttons
3. BoardListDialog	Pop-up window
4. BoardOfLists	Set of lists, label, text-field
5. ConstraintMenu	Menu-bar with pull-down menus
6. DialogContMdf	Empty dialog box
7. DialogContainer	Empty dialog box
8. EditTable	Table and a collection of lists
9. FromList	One main list and a collection of sublists
10. FromListDialog	Pop-up window
11. List	List
12. ListDialog	Pop-up window
13. MainMenu	Set of pushbuttons
14. MessageArea	Area to display messages
15. MessageDialog	Dialog box
16. PromptDialog	Dialog box
17. RadioToggle	Radio box
18. RowOfLabels	Collection of labels
19. RowOfLists	Collection of lists
20. SimpleMessage	Label
21. Table	Table
22. TableDialog	Pop-up window
23. TableFill	Set of label and text-field
24. TableFillDialog	Pop-up window
25. ToggleDialog	Pop-up window

Abstract object classes

a. ActiveListCallData	Supports callback functions
b. BasicComponent [You92]	Parent superclass
c. CustomDialogCallData	Supports callback functions
d. DialogCallbackData	Supports callback functions
e. ListCallbackData	Supports callback functions
f. MainMCallbackData	Supports callback functions
g. MessageManager	Manager of dialog boxes
h. PromptCallbackData	Supports callback functions
i. RadioTCallbackData	Supports callback functions
j. TableFCallbackData	Supports callback functions
k. UIComponent [You92]	Parent superclass

3.4 Description of object classes of RML

Each object class is described using its Component and Inheritance Graphs and a table. The vertices of Component and Inheritance Graphs are Motif widgets or RML classes. When the vertex is an RML class, it is represented by the name of the class surrounded by a rectangular frame. Interpretation of Inheritance Graph is straightforward. A vertex of a Component Graph is composed from the widgets and object classes represented by the vertices of its subgraph. For example Figure 12. shows that ActiveEditTable object class consists of a RowColumn widget (belonging to the Motif Widget Set) that holds an EditTable object and a MainMenu object. The table associated with the object class contains information itemized below:

- **Name** of the object class.
- Parent of the object class (**Superclass**).
- List of other object classes aggregated into the class (**Aggregated classes**).
- Short **Description** of the object class in English.
- List of **Attributes** with short explanation.
- List of **Methods** with short explanation.

In the following, the term "widget" will always refer to a widget defined in the Motif Widget Set. The frequently used phrase "manage the widget" means making it visible on the screen. Technically it involves calling "XtManageChild(widget)" function provided by Xt Intrinsics layer. Manage a component object means to manage its "base widget". "Unmanage" is used analogically.

1.

Name: ActiveEditTable
Superclass: UIComponent
Aggregated classes: EditTable, MainMenu

Description: From the viewpoint of its functionality, this object class is an EditTable object, augmented with pushbuttons. The row of arbitrary number of pushbuttons is placed under the components of EditTable. This has contributed to naming the object ActiveEditTable.

Attributes:

- RowColumn Motif manager widget that lays out the components vertically.
- edTable: An instance of EditTable object class.
- buttons: An instance of MainMenu object class.

Methods:

- Constructor: Header labels of the table are given by a pointer to a constant array. The string format of the header may be specified (NULL if N/A). The content of a message and a label may be specified if needed. There are three constructors available, one for initially empty table, one of each of initial items of table given by constant array and linked list.
- Destructor
- loadMainButtons(): creates "buttons" attribute, (constructor only initializes it). The pushbuttons are specified by a pointer to their labels, type and title of "buttons" are specified as well. See the constructor of MainMenu object class.

(continued on next page)

(continued from previous page)

- getButtons(): accesses "buttons"
- installButtonCallbacks(): installs an external function as callback function for "buttons". See "installMainCallbacks()" method of MainMenu object class.
- getTable(): accesses "edTable"
- access functions: Functions to invoke the methods of "edTable" object.
- manage(): virtual function to manage the object. It manages the components (buttons, edTable) prior to making visible the object on the screen.
- setActSensitive(): sets sensitivity of specified pushbutton of "buttons". See "setSensitive()" method of MainMenu object class.

Component and Inheritance Graphs: Figure 12

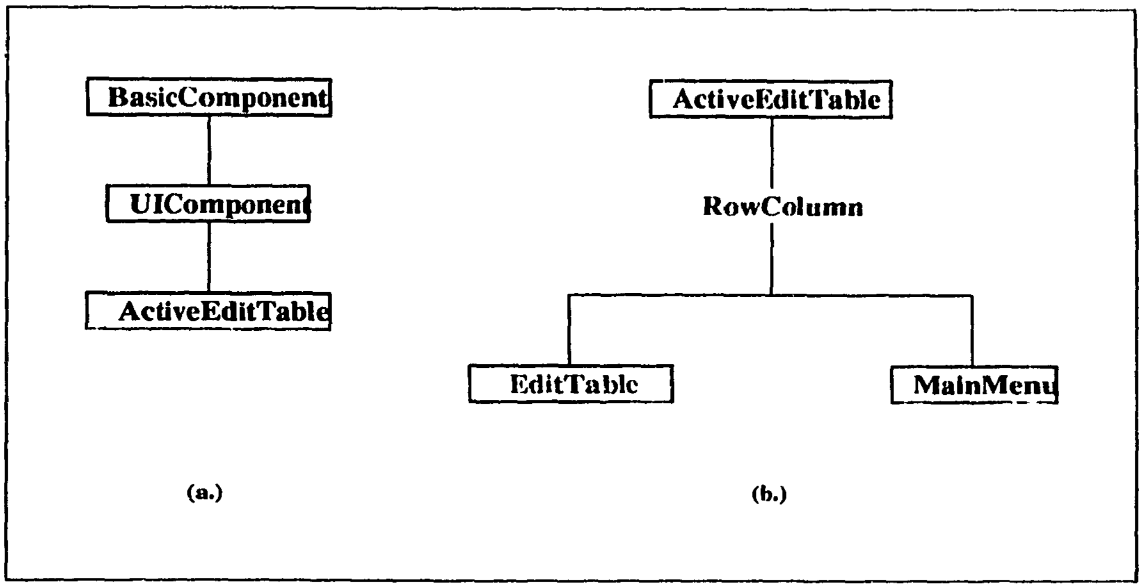


Figure 12: ActiveEditTable object class: (a.)Inheritance Graph (b.)Component Graph

2.

Name: ActiveList
Superclass: List
Aggregated classes: ActiveListCallData, RadioToggle, SimpleMessage

Description: There are two manager widgets prcvided in List object class. ActiveList makes use of the one located under "list" by placing a TextField widget and arbitrary number of togglebuttons (RadioToggle object) into it. These two features provide access to "list". For instance "list" displays some items. Additional items can be added one by one from the TextField. The new item can be appended to or inserted before the selected position(s) of "list", depending on the settings of togglebuttons. Selected positions can be deleted by clicking on the appropriate togglebutton, etc. Two default callback functions are defined as methods in the class, "textDefault" and "toggleD" (see below). The former is invoked upon pressing Enter in the TextField widget, the latter upon clicking on a togglebutton. The default callback functions can be replaced by external functions.

Attributes:

- description: An instance of SimpleMessage object class.
- controls: An instance of RadioToggle object class.
- acd: An instance of ActiveListCallDataobject class.
- textEntry: A TextField motif widget.

Methods:

- Constructor: Selection of the list is given. The content of two messages may be specified (one appears above the "list" of List superclass, the other above "textEntry"). Tcggle buttons are specified giving a pointer to a constant character array of labels of the toggle buttons. There are three constructors available, one for initially empty list, one of each of initial items of list given by constant array and linked list.
- Destructor
- textDefault(): Reads the string entered, and depending on the setting of "controls", adds or inserts it into "list". This virtual function is called upon pressing enter in the TextField widget if no external function has been installed.
- toggleD(): Default callback function of the togglebuttons of "controls". Reads the label of activated ToggleButton. If label is "delete", selected items are deleted from the list. Additional strings can be defined as labels of the togglebuttons of "controls".

(continued on next page)

(continued from previous page)

- installTextCb(): Installs an external function as callback function for "textEntry" Pressing "Enter" in the TextField widget will result invoking the specified function.
- installToggleCb(): Installs the specified external function as callback function for "controls".
- access functions: Functions to invoke the methods of "control" object

Component and Inheritance Graphs: Figure 13.

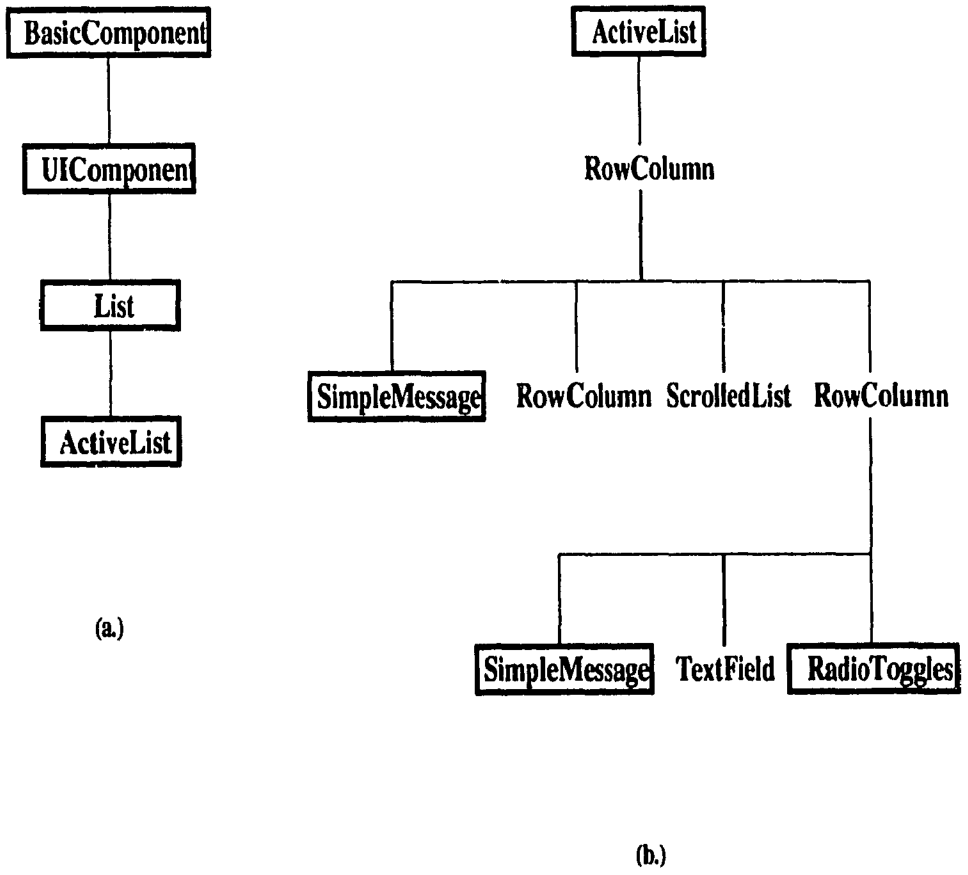


Figure 13: ActiveList object class: (a.) Inheritance Graph and (b.) Component Graph

3.

Name: BoardListDialog
Superclass: DialogContMdf
Aggregated classes: BoardOfLists

Description: Customized dialog box. The action area contains an arbitrary number of push-buttons. The control area of dialog box contains a BoardOfLists object. An example of its use is displaying information (by the means of the components of BoardOfLists object) and pop-down the dialog box upon pressing an "ok" button as acknowledgment.

Attributes:

- boardUnit: an instance of BoardOfLists object class

Methods:

- Constructor: Labels of the action area buttons (See DialogContMdf object class in this section) are given by a pointer to a constant array. Type of action area and a label "titleLabel" (BoardOfLists object class) are also specified.
- Destructor
- clearBoardList(): clears "boardUnit" (See "clearBoard()" method of BoardOfLists object class)
- manage(): virtual function to manage the object.
- getBoardUnit(): returns "boarUnit"

Component and Inheritance Graphs: Figure 14

4.

Name: BoardOfLists
Superclass: UIComponent
Aggregated classes: RowOfLists

Description: This component class consists of a Label widget that belongs to a TextField widget and arbitrary number of List objects (captured by a RowOfLists object). The components are laid out horizontally by a Form manager widget in the sequence Label, TextField, Lists. The components can be surrounded by a Frame widget. The constructor has to be informed if the object is to be created with Frame. New object classes can be derived from BoardOfLists class using the manager widgets of List object class. Any set of components can be aggregated into those areas. BoardOfLists class can be used for browsing and viewing lists.

Attributes:

- type: specifies if there is a frame surrounding the component
- innerForm: holds all the components listed below
- titleText: a TextField motif widget
- titleLabel: a Label motif widget
- choices: an instance of RowOfLists object class

Methods:

- Constructor: "type" and "titleLabel" are given.
- Destructor
- loadBoardList(): Three functions are provided to load lists of "choices". See "loadList1()", "loadList2()" and "loadList3()" methods of RowOfLists object class in this section.
- clearBoard(): clears lists of "choices" and "titleText"
- manage...(): A virtual function to manage the object and functions to manage and unmanage "choices" are provided.
- get....(): a function is provided for each attributes to return them
- set....(): a function is provided for each attributes ("innerForm", "titleText", "titleLabel" and lists of "choices") to set their resources.

Component and Inheritance Graphs: Figure 15

5.

Name: ConstraintMenu
Superclass: UIComponent
Aggregated classes: N/A

Description: This component class consists of a menu-bar with three menu-titles (Implemented using CascadeButton widgets). Each has a pull-down menu of arbitrary number of menu-items. The pull-down menus are created by invoking the member function BuildPullDownMenu()[4]. The menu-items of the pull-down menu can be implemented using PushButton, ToggleButton or Label widgets. All that information (including callback functions) is passed to BuildPullDownMenu().

Attributes:

- commandTitle: CascadeButton widget, its label is a menu title
- optionTitle: CascadeButton widget, its label is a menu title
- helpTitle: CascadeButton widget, its label is a menu title
- commandPulldwn: Stores all relevant information of the pull-down menu associated with "commandTitle".
- optionPulldwn: Stores all relevant information of the pull-down menu associated with "optionTitle".
- helpPulldwn: Stores all relevant information of the pull-down menu associated with "helpTitle".

Methods:

- Constructor: Three pointers to the data structure containing all information necessary, one for each menu title is passed to the constructor. The three menu titles are specified by an array of pointers to character constants.
- Destructor
- BuildPulldownMenu(): Builds pull-down menus (used by the constructor also). Creates the pull-downs, installs callback functions and stores the created menu item widget in "commandPulldwn", "optionPulldwn" and "helpPulldwn".

(continued on next page)

(continued from previous page)

- get....(): a function is provided for each attributes to return them
- setHelp(): Calls the appropriate Motif function to place "helpTitle" to the far right end of the menu-bar.

Component Graph: Figure 16
Inheritance Graph: Figure 17

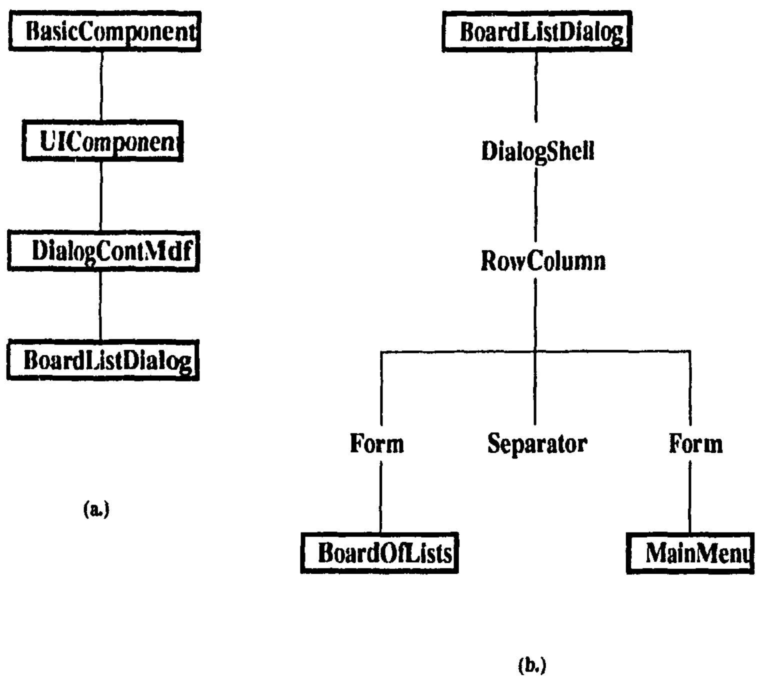


Figure 14: BoardListDialog object class: (a.) Inheritance Graph and (b.) Component Graph

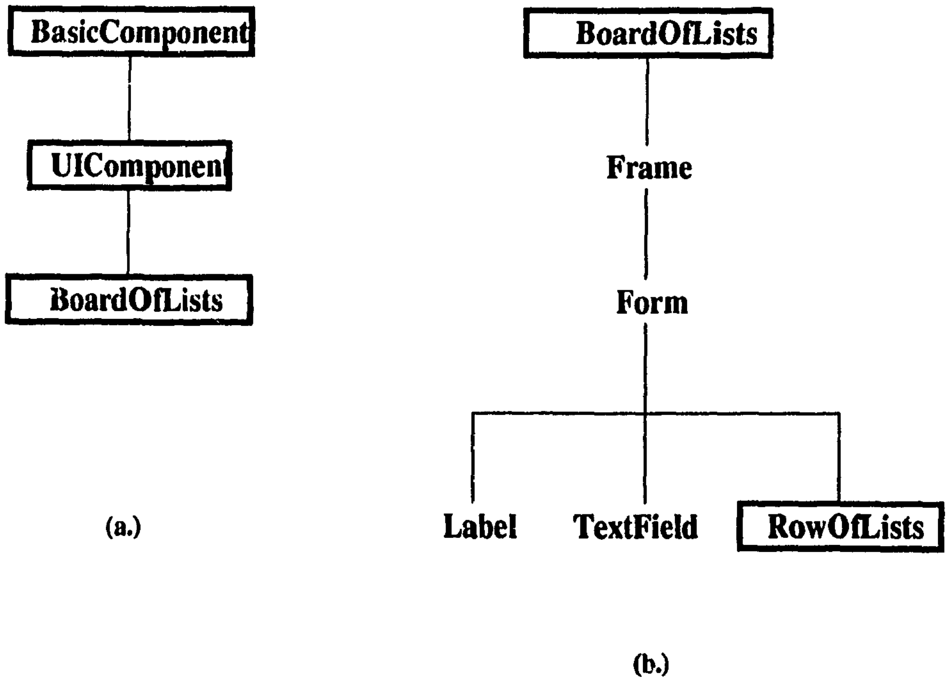


Figure 15: BoardOfLists object class: (a.) Inheritance Graph and (b.) Component Graph

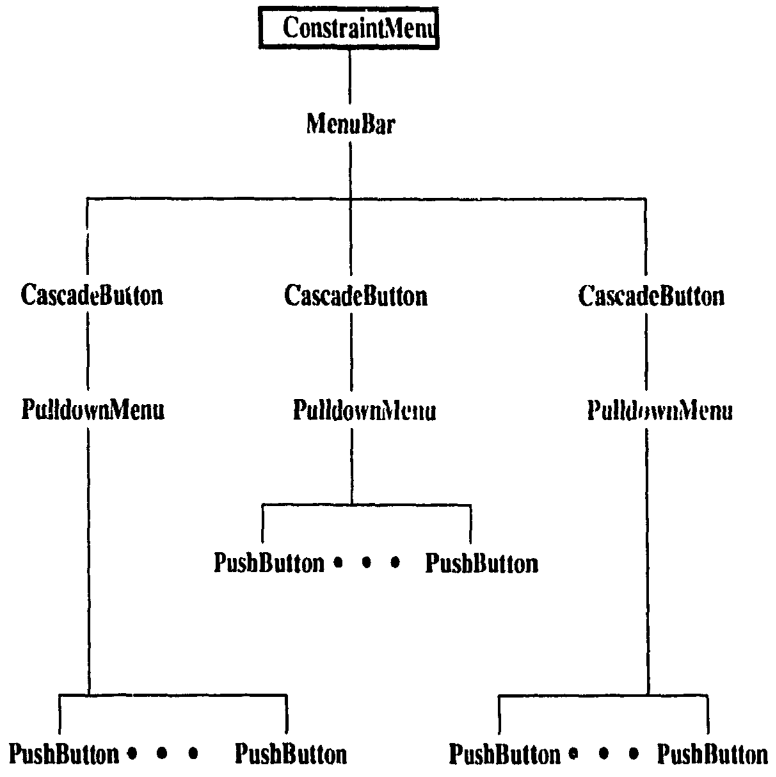


Figure 16: ConstraintMenu object class: Component Graph

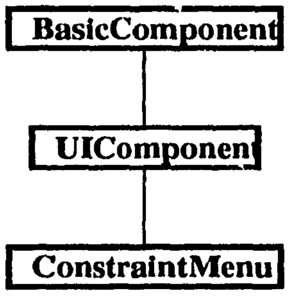


Figure 17: ConstraintMenu object class: Inheritance Graph

6.

Name: DialogContainer
Superclass: UIComponent
Aggregated classes: N/A

Description: This component class is a shell of customized dialog boxes. See descriptions of DialogContMdf object class The control area of DialogContainer is "empty" (contains only the manager widget) Classes can be derived by aggregating other objects or components into the control area. DialogContainer differs from DialogContMdf object class in that action area contains three pushbuttons with "Ok", "Cancel", "Help" labels by default. Labels can be set to arbitrary values and each of the pushbuttons can be unmanaged

Attributes:

- dialogRowcol· RowColumn Motif widget to hold all the components
- ActionForm: Form Motif widget to hold the components of action area.
- ControlForm: Form Motif widget to hold the components of control area.
- separator. Separator Motif widget to divide action and control areas.
- okButton. Pushbutton Motif widget
- cancelButton: Pushbutton Motif widget
- helpButton: Pushbutton Motif widget
- ccd: An instance of CustomDialogCallData object class, used to register the different external callback functions

Methods:

- Constructor: Creates a dialog box with three buttons. An integer value is given to the constructor, to specify button spacing.
- Destructor
- RowColManage(): manage "dialogRowcol"
- ControlFormManage(): manage "ControlForm"
- return functions: Functions are provided to return "ControlForm", "okButton", "cancelButton", "helpButton".

(continued on next page)

(continued from previous page)

- installDialogCallback(): installs an external function as callback function of the pushbuttons of "actionButtons".
- setFocus(). sets focus on the specified widget

Component and Inheritance Graphs: Figure 18

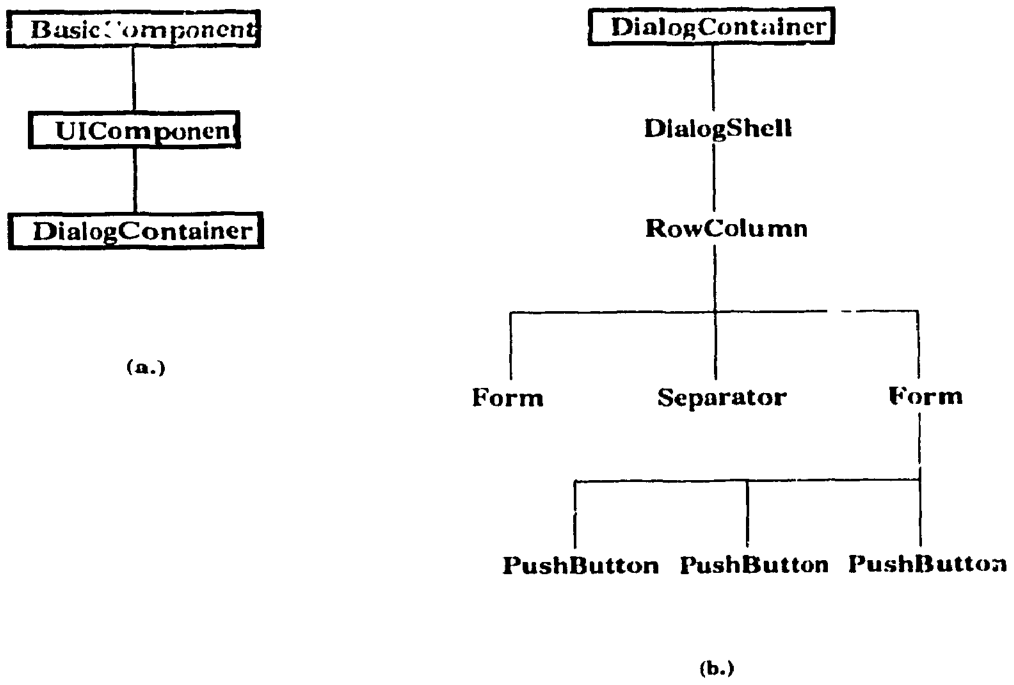


Figure 18: DialogContainer object class: (a.) Inheritance Graph and (b.) Component Graph

7.

Name: DialogContMfd
Superclass: UIComponent
Aggregated classes: MainMenu

Description: This component class is the shell of customized dialog boxes. The dialog box is divided into two parts, control and action areas, by a separator widget[Fou93]. The components composing the control area provide the functionality of the dialog. The action area contains pushbuttons whose callback functions initiate "the action of the dialog box". The control area of DialogContMdf is "empty" (contains only the manager widget). Classes can be derived by aggregating other objects or components into the control area. The action area contains arbitrary number of pushbuttons.

Attributes:

- dialogRowcol: RowColumn Motif widget to hold all the components
- ActionForm: Form Motif widget to hold the components of action area.
- actionButtons: An instance of MainMenu object class. (Pushbuttons of the dialog)
- ControlForm: Form Motif widget to hold the components of control area.
- separator: Separator Motif widget to divide action and control areas.

Methods:

- Constructor:Type and labels of "actionButtons" are given to the constructor.
- Destructor
- RowColManage(): manage "dialogRowcol"
- ControlFormManage(): manage "ControlForm"
- getActButtons(): returns "actionButtons"
- ControlWidget(): returns "ControlForm"
- installDialogCallback(): installs an external function as callback function of the pushbuttons of "actionButtons".

Component and Inheritance Graphs: Figure 19

8.

Name: EditTable
Superclass: Table
Aggregated classes: BoardOfLists

Description: This component class aggregates a BoardOfLists object into the manager widget (located under "list") of Table object class. This results in a Table augmented vertically with an area containing a Label, TextField and arbitrary number of List objects. The components of BoardOfLists are held together by a Frame.

Attributes:

- board: An instance of BoardOfLists object class.

Methods:

- Constructor: There are three constructors available, as in the case of Table object class. One additional parameter is given for "titleLabel" of "board".
- Destructor
- loadEdTableList(): Three functions are provided to load list of "board". See "loadList1()", "loadList2()" and "loadList3()" methods of RowOfLists object class
- manage(): methods are provided for managing and unmanaging "board".
- clearEdTable(): clears the "list" (Table class), and "board".
- access functions: functions to invoke the methods of "board".

Component and Inheritance Graphs: Figure 20

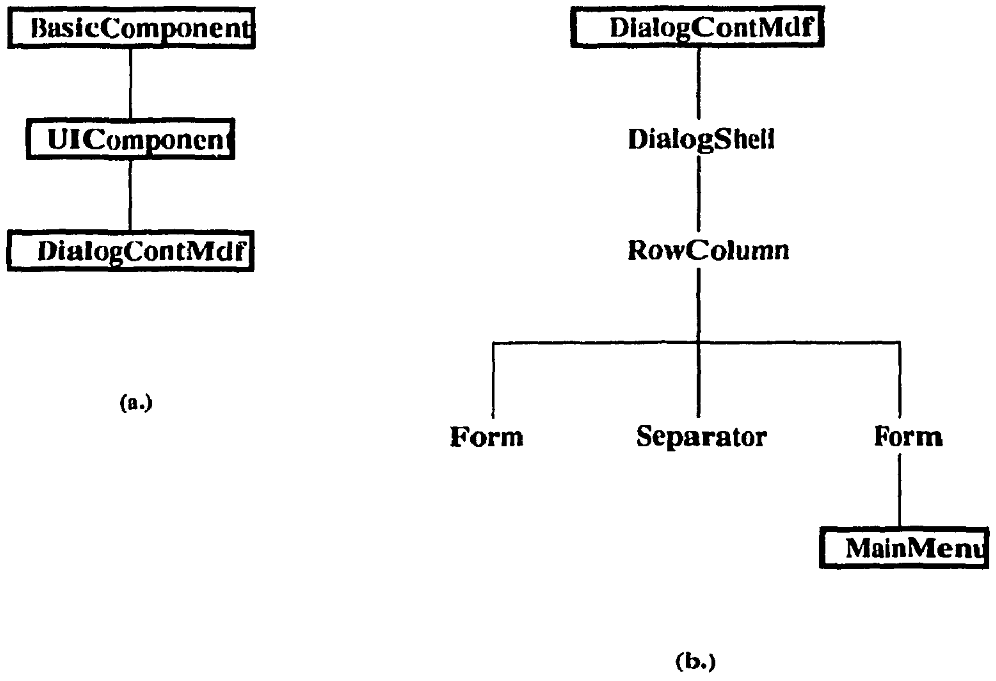


Figure 19: DialogContMdf object class: (a.)Inheritance graph and (b.)Component graph

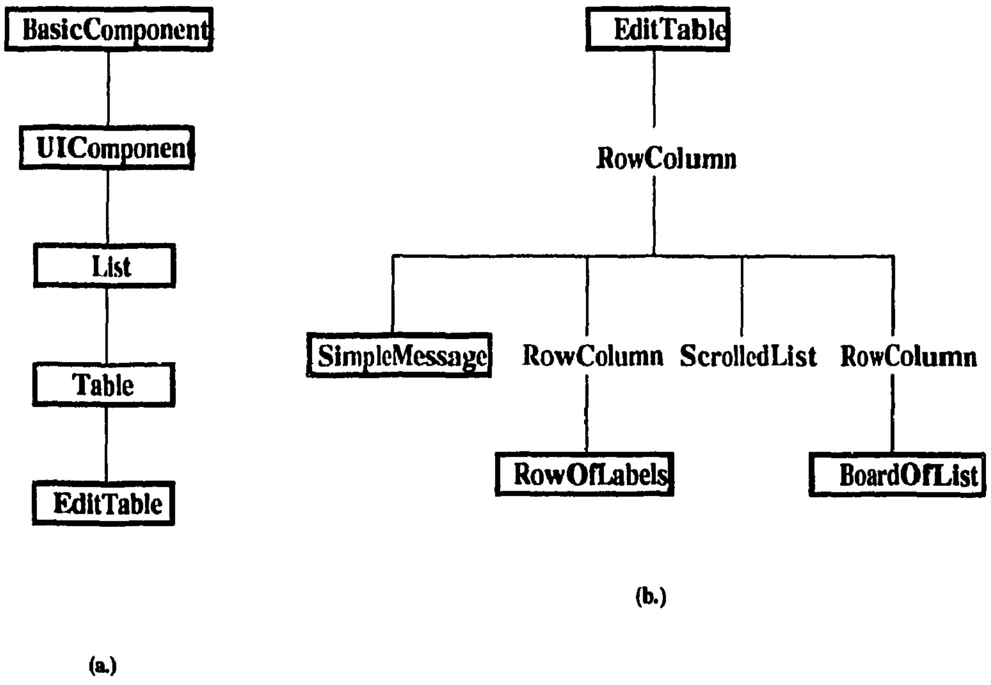


Figure 20: EditTable object class: (a.)Inheritance Graph and (b.)Component Graph

9.

Name: FromList
Superclass: List
Aggregated classes: RowOfLists

Description: This component class makes use of the manager widget located under "list" of a List object by aggregating a RowOfLists object into it.This gives the following arrangement. There is a main list and underneath a row of arbitrary number of lists ("fromList"). The idea behind this class is that the main list can be constructed using "fromList". An example is BuildListDefault() member function described below.

Attributes:

- fromList: an instance of RowOfLists object class

Methods:

- Constructor: There are three constructors identical to those given for List object class.
- Destructor
- loadFromList(): Three functions are provided to load the lists of "fromlist". See "loadList1()", "loadList2()" and "loadList3()" methods of RowOfLists object class
- manage(): functions to manage and unmanage "fromList"
- getFromLists(): returns "fromlist"
- buildListDefault(): Builds list from the lists of "fromList" if it consists of three lists (list1, list2, list3) by concatenating one selected item from each list. list1 can be multiple or single select, list2 and list3 single select. The format of the resulting string is the only parameter of the function.

Component and Inheritance Graphs: Figure 21

10.

Name: FromListDialog
Superclass: DialogContMdf
Aggregated classes: FromList

Description: Customized dialog box. The action area (of superclass DialogContMdf) may contain arbitrary number of pushbuttons. The control area of the dialog box contains a FromList object.

Attributes:

- worklists: An instance of FromList object class.

Methods:

- Constructor: There are three constructors available, as in the case of FormList object class. There is one additional parameter to specify the type of pushbuttons (MainMenu object) in DialogContMdf class.
- Destructor
- loadFromListD(): Three functions are provided to load the lists of"worklists". See "loadList1()", "loadList2()" and "loadList3()" methods of RowOfLists object class
- manage(): Functions provided to manage and unmanage "workLists" and its components.
- buildFromListDefault(): invoke buildListDefault() function of "workLists".
- getWorkLists(): returns "worklists"
- access functions: functions to invoke the methods of "woklists"

Component and Inheritance Graphs: Figure 22

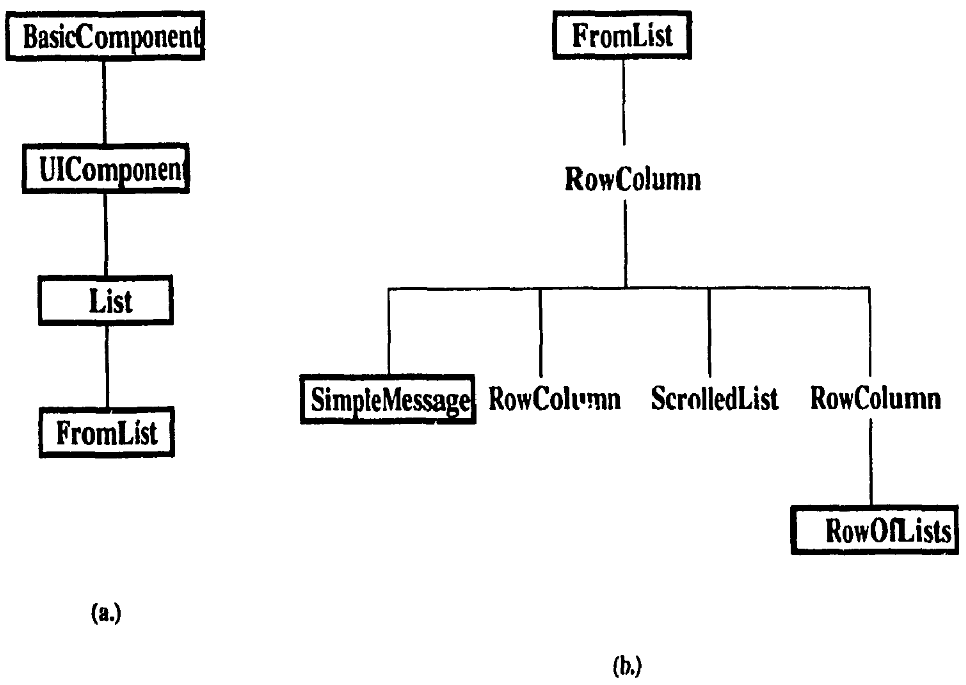


Figure 21: FromList object class: (a.)Inheritance Graph and (b.)Component Graph

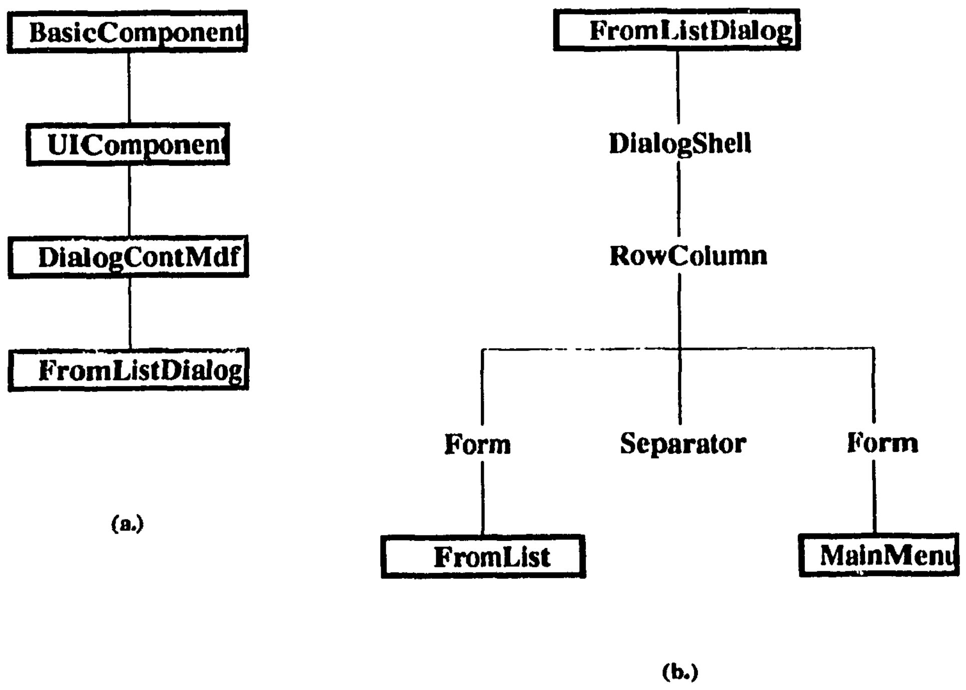


Figure 22: FromListDialog object class: (a.)Inheritance Graph and (b.)Component Graph

11.

Name: List
Superclass: UIComponent
Aggregated classes: ListCallbackData, SimpleMessage

Description: List component object class consists of the following components laid out vertically:

- An "empty" manager widget (RowColumn) where derived classes can hook up.
- A character string that can be a short instruction, explanation or title of the "list".
- A scroll-able list widget.
- An "empty" RowColumn manager widget (under "list") where derived classes can hook up.

The methods of List involve operations on selected items, addition, deletion etc.

Attributes:

- topExtendCol: RowColumn motif widget located above "list" to which derived classes can hook up
- instruction: An instance of SimpleMessage object class.
- list: scroll-able list, Motif widget
- lcd: An instance of ListCallbackData object class.
- extendCol: RowColumn widget located under "list" to which derived classes can hook up

Methods:

- Constructor: There are three constructors available, one for initially empty table, one of each of initial items of table given by constant array and linked list.
- Destructor
- installListCb(): installs a given external function as callback. The external function will be invoked upon selection of items in "list".
- manage(): Functions are provided to manage and unmanage "topExtendCol" and "extendCol".
- get....(): Functions that return the attributes "topExtendCol", "extendCol", "list".

(continued on next page)

(continued from previous page)

- getItems(): Returns the list of items of "list".
- getSelectPos( ): Prepares a list of positions of selected items.
- getSelectedItems(): Returns list of selected items of " list".
- addItem(): Appends the given item (character string) to the "list".
- addItems(): Appends the given items to the "list". (items are given by a linked list)
- insertItem(): Insert the given item before the selected position.
- selectedDelete(): Delete items of "list" at the selected position(s).
- clearList(): empty "list"
- setResources(): Functions to provide accesses to various resources of "list".

Component and Inheritance Graphs: Figure 23

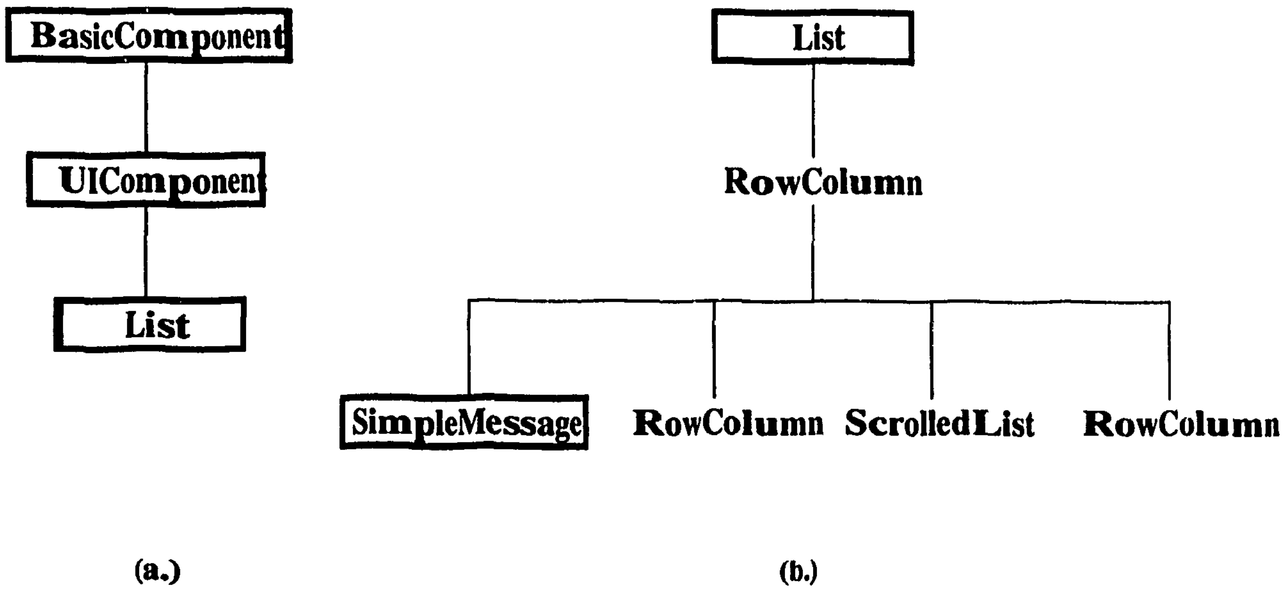


Figure 23: List object class: (a.)Inheritance Graph and (b.)Component Graph

12.

Name: ListDialog
Superclass: DialogContMdf
Aggregated classes: ActiveList

Description: Customized dialog box. The action area (of superclass DialogContMdf) may contain arbitrary number of pushbuttons. The control area of the dialog box contains an ActiveList object.

Attributes:

- listUnit: An instance of ActiveList object class

Methods:

- Constructor: Three constructors are available as in the case of ActiveList class. There is one additional parameter here, the action area type specifier (pushbuttons of DialogContMdf).
- Destructor
- manage(): Manage and unmanage components of "listUnit"
- install...(): Functions to install callback functions of "listunit".
- access functions: functions to invoke the methods of "listUnit" involving accessing items of "list" (List object class).

Component and Inheritance Graphs: Figure 24

13.

Name: MainMenu
Superclass: UIComponent
Aggregated classes: N/A

Description: This component class contains an arbitrary number of pushbuttons laid out horizontally or vertically. The pushbuttons may be surrounded by a frame and have a common title. The pushbuttons are defined by their labels that are supplied to the constructor. Activating any of the pushbuttons results invoking the default or previously installed external function.

Attributes:

- rowcol: RowColumn Motif widget to hold all the components.
- pushButton: Array of pushbutton widgets.
- mmc: An instance of MainMCallbackData object class, to register the external function and other data to facilitate communication between callback functions and calling objects.
- type: Integer variable, type = 0 means there is no frame surrounding "pushButtons".
- buttonPushed: Stores "pushButton[i]" that has been activated more recently.

Methods:

- Constructor: Pushbuttons are specified by their labels that are passed to the constructor using an array of pointers to character. The title and type of the pushbuttons are also specified.
- Destructor
- installMainCallbacks(): Installs external function as callback function.
- getLabelPushed(): Returns the label of most recently activated pushbutton, "buttonPushed".
- get....(): Returns attributes "buttonPushed", "mmc", "pushButton".
- set...(): Functions to set resources of components such as sensitivity and arrangement of "pushButtons", various resources of " w".
- manageRowCol(): manage "rowcol"

Component and Inheritance Graphs: Figure 25

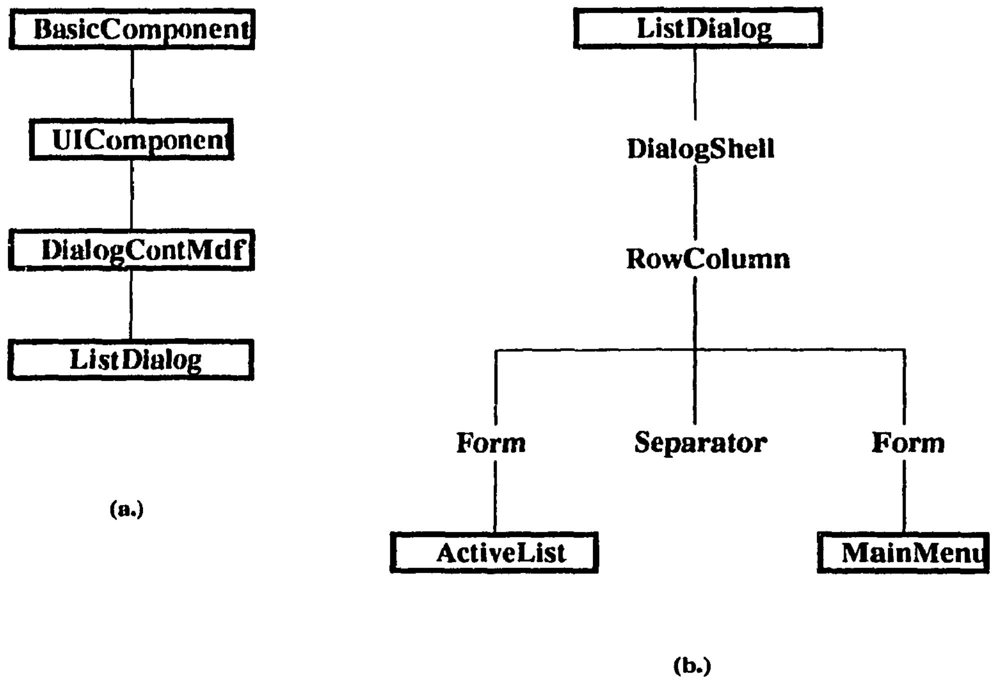


Figure 24: ListDialog object class: (a.)Inheritance Graph and (b.)Component Graph

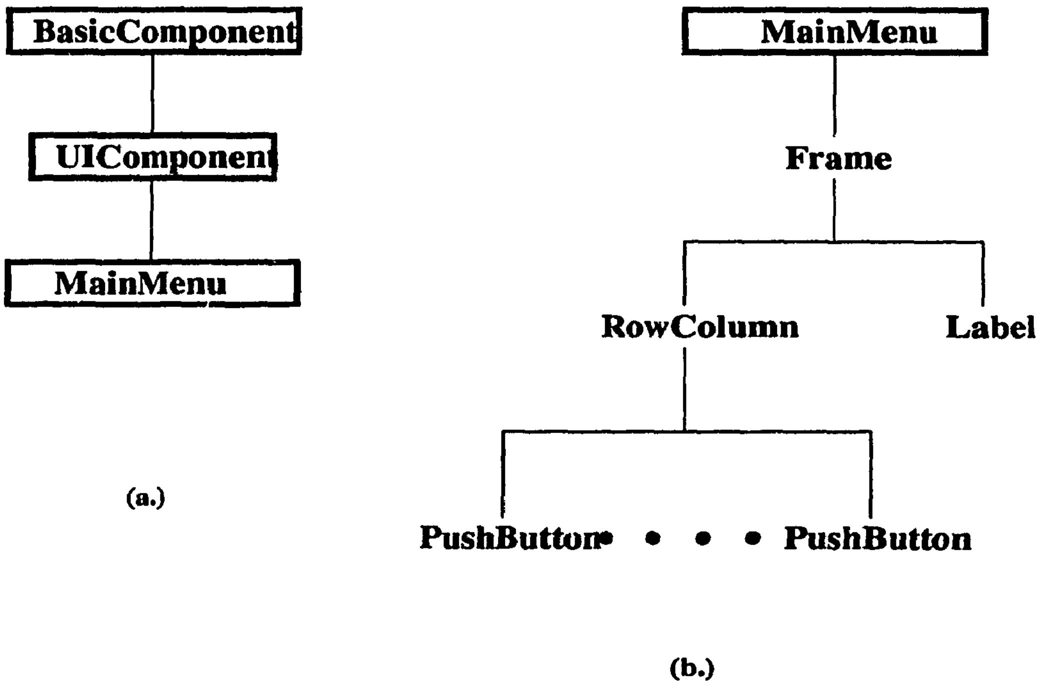


Figure 25: MainMenu object class: (a.)Inheritance Graph and (b.)Component Graph

14.

Name: MessageArea
Superclass: UIComponent
Aggregated classes: N/A

Description: The functionality of this component class is given by a Scroll-able Text widget. The visible number of rows and columns are specified. By default the Text widget is not edit-able, character strings can be displayed one after the other. The most recently displayed character string is highlighted. Applications can use it to display messages to the user.

Attributes:

- textOutput: Scroll-able Text widget. Not edit-able.
- lastLenght: Length of most recently displayed message.

Methods:

- Constructor: The visible number of rows and columns are provided.
- Destructor
- messagePrint(): Prints the given character string into the Text widget.
- composeMessage(): Composes a string from numerous character strings according to the given format and displays it in the Text widget.
- clearArea(): Clears message area.
- setResources(): Visible number of rows and columns, horizontal scrollbar displayed or not, wrap the text in the visible area or not, edit mode (single or multiple lines) can be specified.
- setHighLight(): Most recently displayed character string (message) is highlighted or not can be specified. (default:highlighted)

Component and Inheritance Graphs: Figure 26

15.

Name: MessageDialog
Superclass: BasicComponent
Aggregated classes: DialogCallbackData

Description: The only component of this class is the built in Motif dialog box, MessageDialog. The type of MessageDialog can be any of the eight Motif MessageDialogs. Methods are provided to set the action area buttons, the message and type of the dialog.

Attributes:

- dcb: An instance of DialogCallbackData object class
- dialog: A built in message dialog box

Methods:

- Constructor: The type of Message dialog-box and the message to display have to be provided for the constructor.
- Destructor
- installCallback(): Installs external functions as callback functions. There may be three functions given one for each "ok", "cancel", "help" buttons.
- cleanCallback(): Remove callback functions from object. This is to prepare its reuse.
- get....(): Functions to return pushbuttons of message dialog-box ("ok , "cancel", "help").
- setok(): Manages or unmanages "ok" button of dialog-box. Sets label of button.
- setcancel(): Manages or unmanages "cancel" button of dialog-box Sets label of button.
- sethelp(). Manages or unmanages "help" button of dialog box. Sets label of button.
- setType(): Sets type of dialog box.
- setMessage(): Sets message of dialog box.
- post(): Displays (pops-up) dialog box.

Component and Inheritance Graphs: Figure 27

16.

Name: PromptDialog
Superclass: UIComponent
Aggregated classes: PromptCallbackData

Description: The only component of this class is the built in Motif dialog box, PromptDialog Methods are provided to set the action area buttons, the message and type of the dialog.

Attributes:

- textString: content of textfield
- fileContent: A character string used here as storage of the content of a file.
- pcb: An instance of PromptCallbackData object class.

Methods:

- Constructor: The the textfield label of the prompt dialog-box is given to the constructor.
- Destructor
- installCallback(): Installs external functions as callback functions. There may be four functions given one for each "ok", "cancel", "help" buttons and the text-field.
- loadText(). Gets text entered into the textfield of prompt dialog box.
- loadFilecontent(): This member function is a utility function. It assumes "textString" to be a filename. Attempts to open the file and reads in its content into "fileContents". Initiates error message if the file does not exist or can not be opened.
- setok(): Manages or unmanages "ok" button of dialog box. Sets label of button.
- setcancel(): Manages or unmanages "cancel" button of dialog-box. Sets label of button.
- sethelp(): Manages or unmanages "help" button of dialog - box.Sets label of button.
- get....(): Functions are provided to return "textString", "fileContents", "pcb"and the push-buttons of dialog box.
- post(): Displays (pops-up) dialog box.

Component and Inheritance Graph: Figure 28

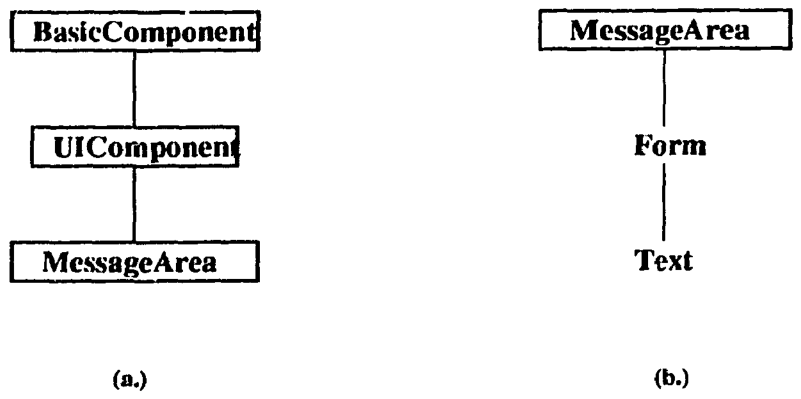


Figure 26: MessageArea object class: (a.)Inheritance Graph and (b.)Component Graph

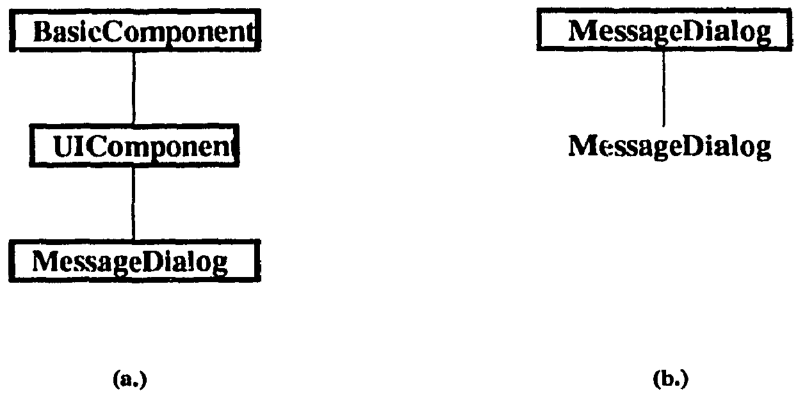


Figure 27: MessageDialog object class: (a.)Inheritance Graph and (b.)Component Graph

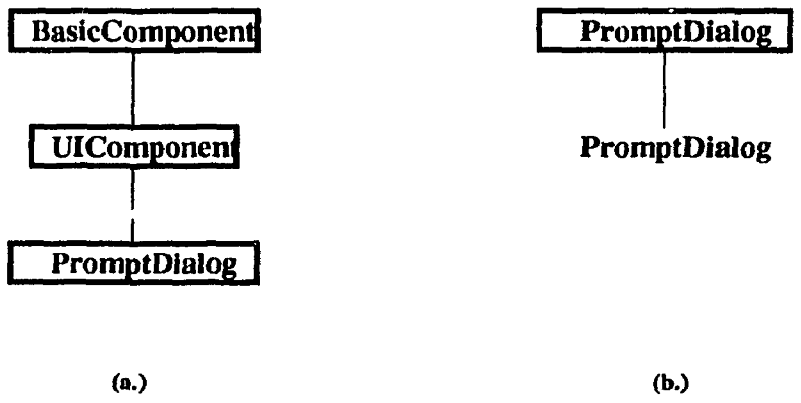


Figure 28: PromptDialog object class: (a.)Inheritance Graph and (b.)Component Graph

17.

Name: RadioToggle
Superclass: UIComponent
Aggregated classes: RadioTCallbackData

Description: This component class contains an arbitrary number of togglebuttons defined by their labels. The (row * column) arrangement of the toggles can be defined. Activating any of the togglebuttons results invoking the default or a previously installed external function.

Attributes:

- Labels: Labels of ToggleButtons
- rtc: An instance of RadioTCallbackData object class. Stores relevant information of external callback functions of ToggleButtons.
- buttonSet: ToggleButton that has been set most recently.

Methods:

- Constructor: Pointer to ToggleButton labels are supplied.
- Destructor
- installCallbacks(): Installs external functions as callbacks.
- getButtonSet(): Returns "buttonSet"
- getLabelSet(): Returns the label of most recently activated ToggleButton.
- getCallData(): Returns "rtc".
- clearToggles(): Sets the state of all toggles to "0".
- setPackColOr(): Specifies arrangement of toggles through resources. (sets number of columns or rows, orientation, packing)

Component and Inheritance Graphs: Figure 36

18.

Name: RowOfLabels
Superclass: UIComponent
Aggregated classes: N/A

Description: This component class consists of arbitrary number of Labels. An example of its use is the table header.

Attributes: None

Methods:

- Constructor: Two constructors are developed for th'ɔ class. The array of Labels is passed to both constructor. One constructor creates the labels and prints them one after the other without formatting. The other constructor takes a character string parameter interprets it as the format of the labels and print them accordingly.
- Destructor

Component and Inheritance Graphs: Figure 29

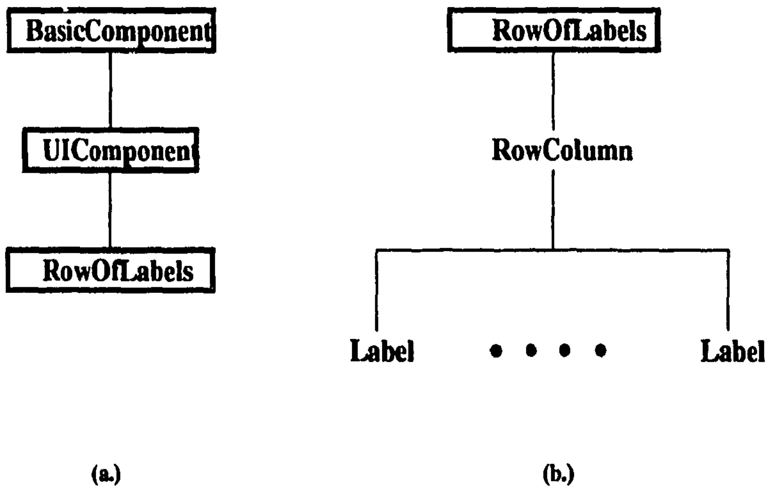


Figure 29: RowOfLabels object class: (a.)Inheritance Graph and (b.)Component Graph

Name: RowOfLists
Superclass: UIComponent
Aggregated classes: List

Description: This component class consists of arbitrary number of List objects laid out horizontally.

Attributes:

- lists: Array of List objects.

Methods:

- Constructor: The usual parent and name the only parameters the constructor needs. The constructor creates a RowColumn widget to hold the List objects, that are created and loaded later on using "loadList1()", "loadList2()", "loadList3()" methods below.
- Destructor
- loadList1(): Loads a List object into the RowColumn widget "w" created by constructor. The List object is created inside "loadList1()" function taking the constructor of List that is for "initial items of "list" given by constant array".
- loadlist2(): Loads a List object into the RowColumn widget "w" created by constructor. The List object is created inside "loadList2()" function taking the constructor of List that is for "initial items of "list" given by a linked list".
- loadlist3(): Loads a List object into the RowColumn widget " w" created by constructor. The List object is created inside "loadList2()" function taking the constructor of List that is for empty "list".
- getList(): Returns the specified list, "lists[i]".
- getSelectItem(): Returns the list of selected items of the specified list, "lists[i]".
- getItems(): Returns the list of items of the specified list, "lists[i]".
- addItem(): Appends item to specified list, "lists[i]".
- addItems(): Appends items to specified list, "lists[i]".

(continued on next page)

(continued from previous page)

- insertItem(): Inserts item into specified list, "list[i]" before selected position.
- selectedDelete(): Deletes selected item(s) from specified list, "list[i]".
- clearList(): Deletes all items (empties list) from specified list "list[i]".
- clearLists(): Deletes all items from each lists. (Empties "lists[0] .. lists[MAX]")
- setResources(): Sets resources of specified list. (Type of selection, Number of visible items)

Component and Inheritance Graphs: Figure 30

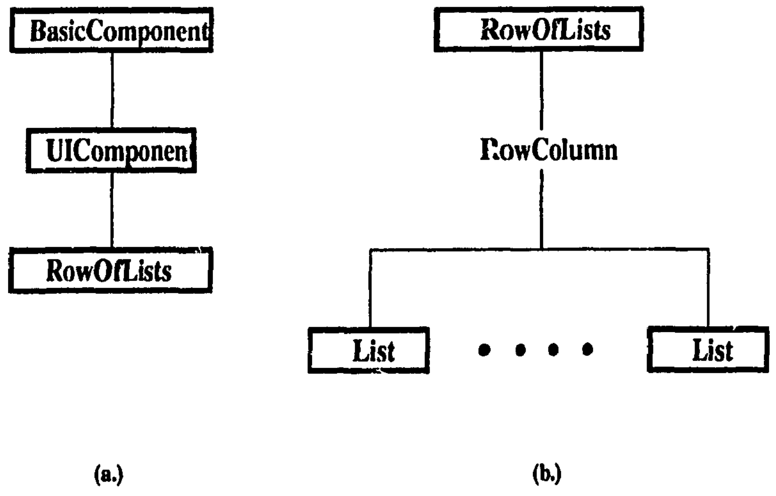


Figure 30: RowOfLists object class: (a.)Inheritance Graph and (b.)Component Graph

20.

Name: SimpleMessage
Superclass: UIComponent
Aggregated classes: N/A

Description: Component class that creates and displays a short message (text) as a Label widget.

Attributes:

- message: Label widget

Methods:

- Constructor: The content of the message is provided to the constructed.
- Destructor
- setmessage(): Sets the text (content) of the message.

Component and Inheritance Graphs: Figure 31

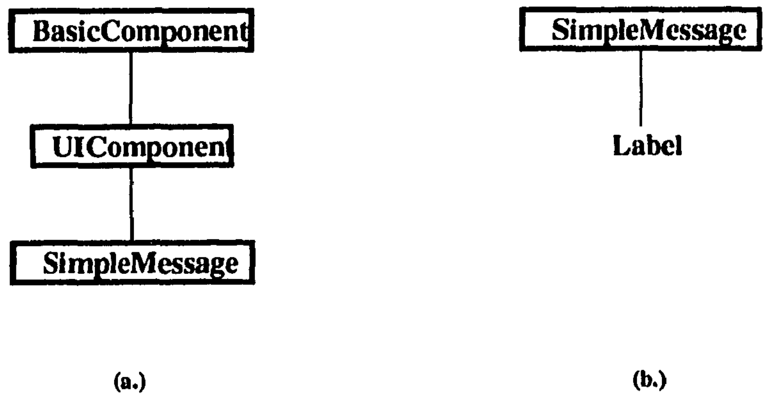


Figure 31: SimpleMessage object class: (a.)Inheritance Graph and (b.)Component Graph

21.

Name: Table
Superclass: List
Aggregated classes: RowOfLabel

Description: Table object class aggregates a RowOfLabels objects into the manager RowColumn widget (located above "list") provided by List object class. RowOfLabel object acts as table header.

Attributes:

- tableHeader: An instance of RowOfLabel object class.

Methods:

- Constructor: Three constructors are available identical to the Constructors of List, augmented with the parameters to set up RowOfLabels. Namely array of labels, number of labels and format.
- Destructor

Component and Inheritance Graphs: Figure 32

22.

Name: TableDialog
Superclass: DialogContMdf
Aggregated classes: Table

Description: Customized dialog box. The action area (of superclass DialogContMdf) may contain arbitrary number of pushbuttons. The control area of the dialog box contains a Table object.

Attributes:

- table: An instance of Table object class.

Methods:

- Constructor: Three constructors are available identical to constructors of Table, augmented with the parameters to set up DialogContMdf. Namely type of action area (type of MainMenu), array of labels of the pushbuttons and number of pushbuttons.
- Destructor
- getTable(): Returns "table".
- clearTable(): Deletes all items of "table".

Component and Inheritance Graphs: Figure 33

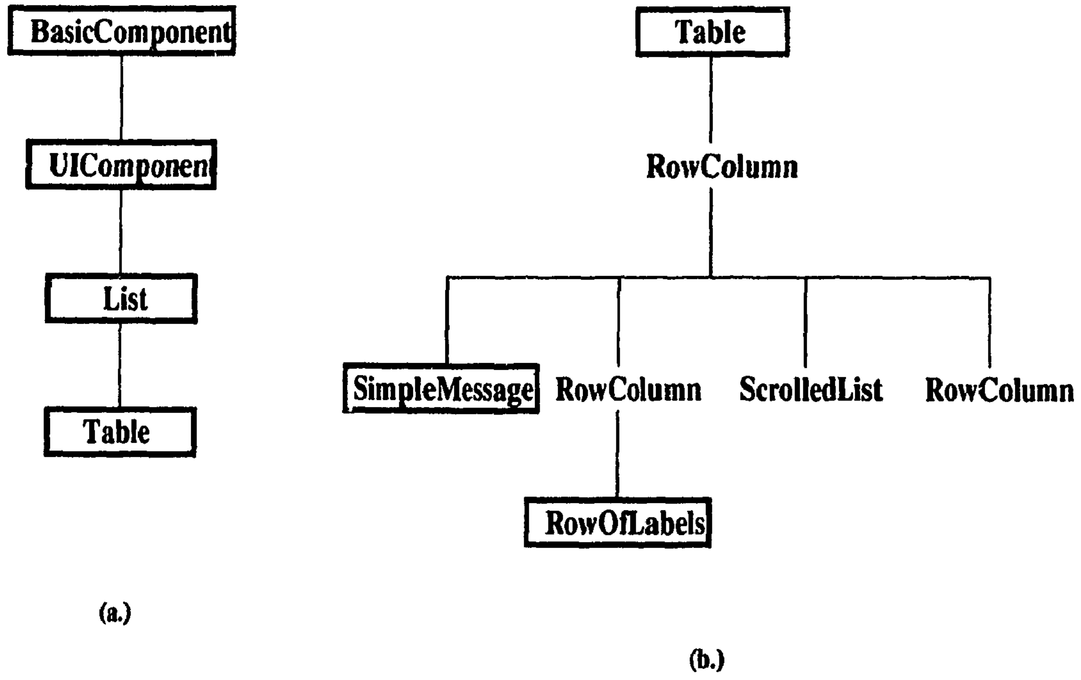


Figure 32: Table object class: (a.)Inheritance Graph and (b.)Component Graph

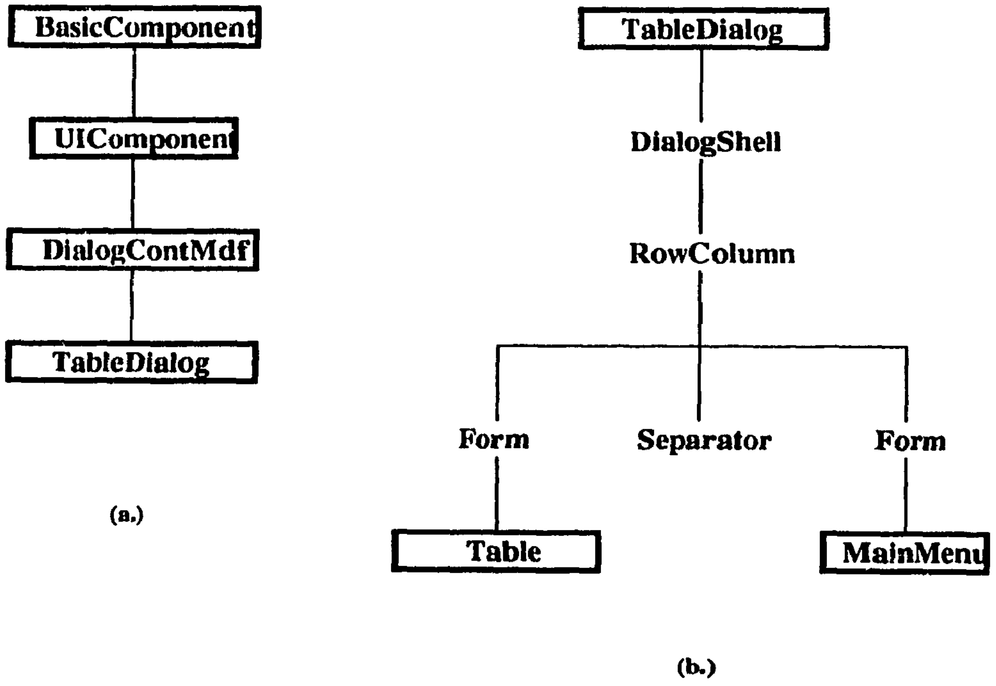


Figure 33: TableDialog object class: (a.)Inheritance Graph and (b.)Component Graph

23.

Name: TableFill
Superclass: UIComponent
Aggregated classes: TableFCallbackData

Description: Component class that consists of arbitrary number of pairs of Label and TextField widgets laid out vertically. Pressing Enter in any of the TextField widget will result in invoking the textDefault() member function or external function if there has been installed one.

Attributes:

- fillForm: Form motif widget to hold together the corresponding text and label.
- fillText: Array of text-fields.
- fillLabel: Array of labels associated with elements of "fillText".
- tfc: An instance of TableFCallbackData object class.

Methods:

- Constructor: Array of ıabels and number of labels are specified.
- Destructor
- installCallbacks(): Installs external functions as callbacks of "fillText".
- getTextValues(): Returns list of text values (contents of TextField widgets).
- setTextValues(): Sets text field value.
- clearTextFields(): Clears all elements of "fillText".

Component and Inheritance Graphs: Figure 34

24.

Name: TableFillDialog
Superclass: DialogContMdf
Aggregated classes: TableFill

Description: Customized dialog box. The action area (of superclass DialogContMdf) may contain arbitrary number of pushbuttons. The control area of the dialog box contains a TableFill object.

Attributes:

- table: An instance of TableFill object class .

Methods:

- Constructor: Action area type, labels of action area buttons (MainMenu), array of labels and number of labels (TableFill) are specified.
- Destructor
- installTableTextCb(): Installs external functions as callbacks of "fillText" of TableFill.
- getTable(): Returns "table".
- getTableTextValues(): Returns list of text values of "fillText" of "table" (TableFill).
- setTableTextValues(): Sets text values of "fillText" of "table" (TableFill).
- clearTableTextFields(): Clears all elements of "fillText" "table" (TableFill).

Component and Inheritance Graphs: Figure 35

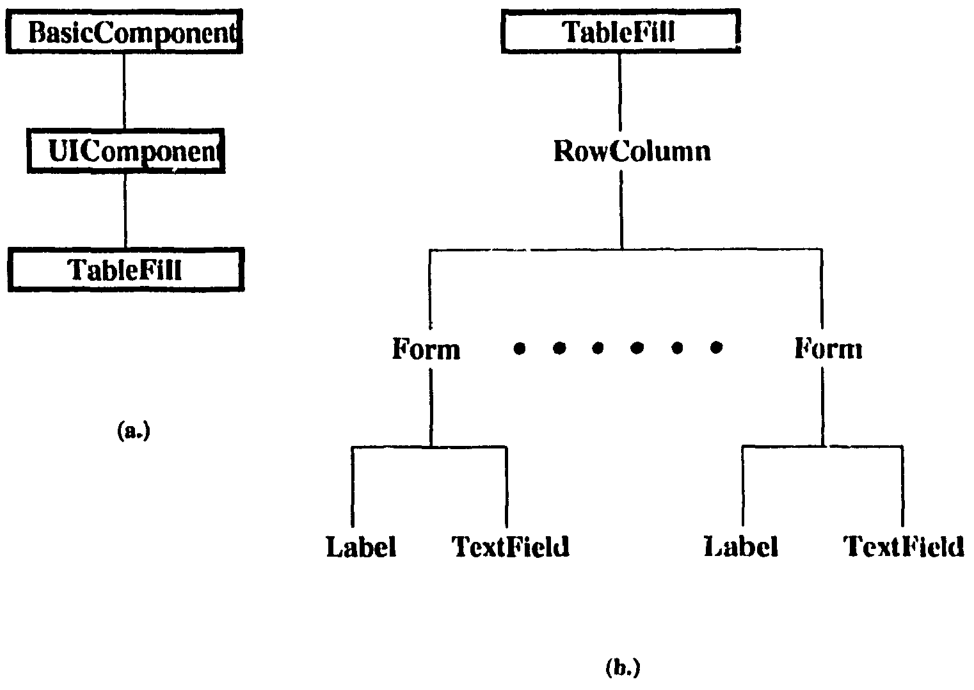


Figure 34: TableFill object class: (a.)Inheritance Graph and (b.)Component Graph

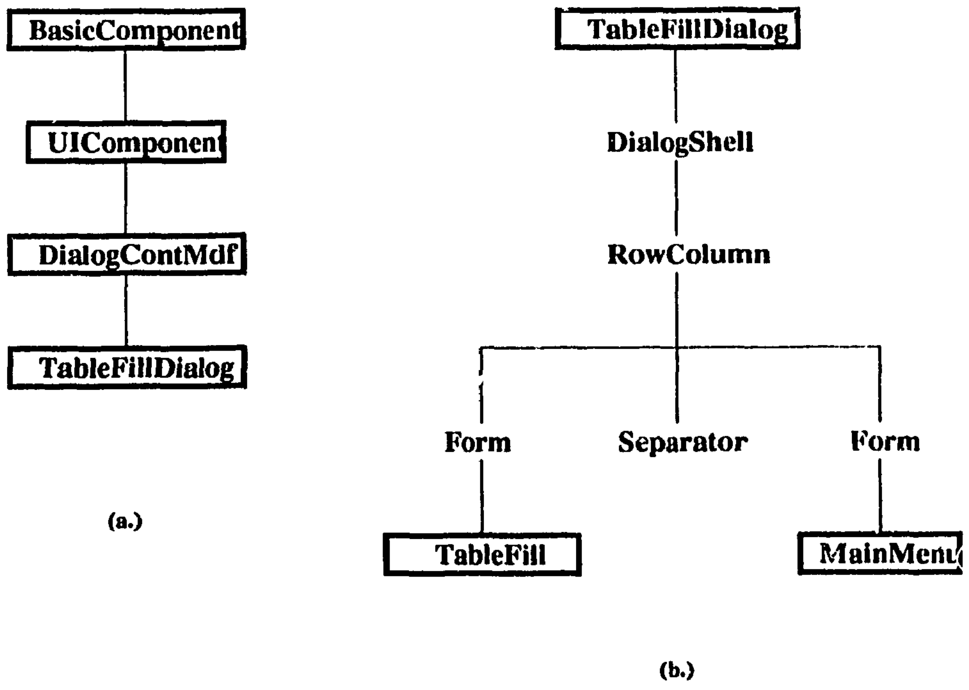


Figure 35: TableFillDialog object class: (a.)Inheritance Graph and (b.)Component Graph

25.

Name: ToggleDialog
Superclass: DialogContMdf
Aggregated classes: RadioToggle, SimpleMessage

Description: Customized dialog box. The action area contains an arbitrary number of pushbuttons up to a maximum value which is set in MainMenu object class. The control area of the dialog box may contain a short message and a set of toggle buttons. Hence the control area is an aggregate of SimpleMessage and RadioToggle object classes. The callback functions of the toggle buttons and action area push buttons can be specified by any external function.

Attributes:

- w: RowColumn Motif manager widget that lays out the components vertically.
- message: An instance of SimpleMessage object class.
- toggleBox: An instance of RadioToggle object class.

Methods:

- Constructor: Labels of the action area buttons are given by a pointer to a constant array. The content of the message is specified if there exist one. Toggle buttons are specified giving a pointer to a constant character array of labels of the toggle buttons.
- Destructor
- installToggleCb(): installs an external functions as callback function
- getToggleSet(): returns the togglebutton which is set
- getLabelSet(): returns label of toggle button which is set
- setMessage(): set message
- clearDialogToggles(): set all toggles to '0', hence clear them

Component and Inheritance Graphs: Figure 37

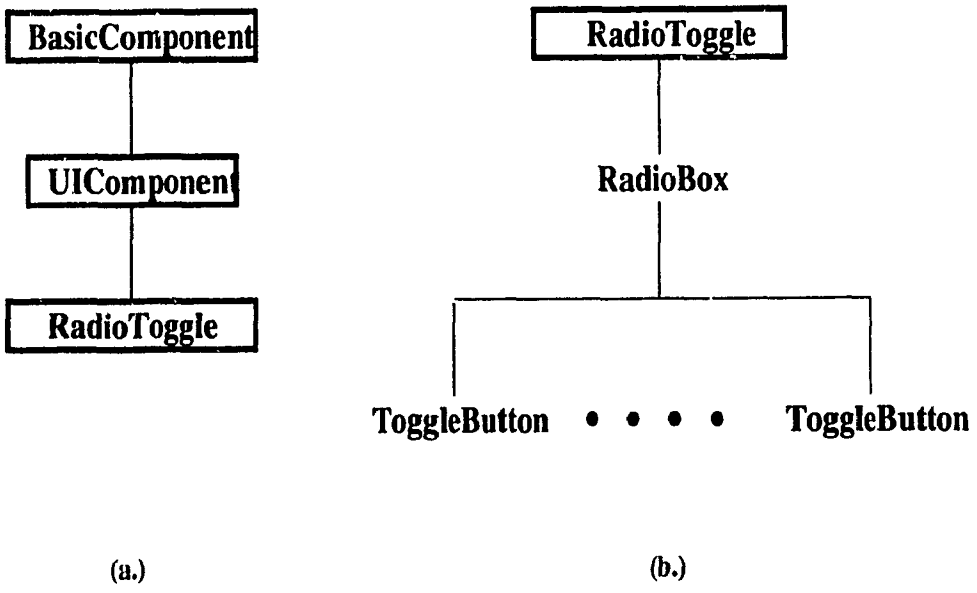


Figure 36: RadioToggle object class: (a.)Inheritance Graph and (b.)Component Graph

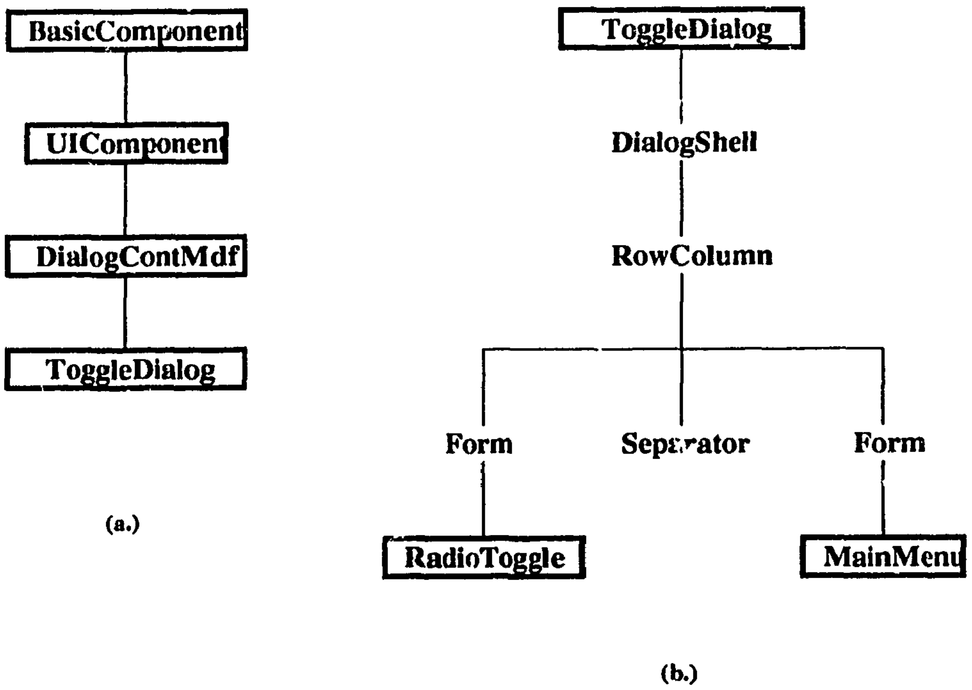


Figure 37: ToggleDialog object class: (a.)Inheritance Graph and (b.)Component Graph

a.

Name: ActiveListCallData
Superclass: N/A
Aggregated classes: N/A

Description: Abstract object class. This is an auxiliary class to ActiveList object class. It stores all relevant data to invoke an external function "f" instead of the default "textDefault" function that is a member function of ActiveList.

Attributes:

- activeList: pointer to an instance of ActiveList object class
- text: pointer to a function which takes one parameter, (void *). This external function is the callback function of "textEntry" (See the description of ActiveList object class in this section.)
- clientData1, clientData2: pointers to any data structure or object to facilitate the information exchange between the object and external callback function.

Methods:

- Constructor: Takes all four attributes to create an instance of the object class.
- Destructor
- return functions: A function is provided for each attribute to return them
- setClientData1(): set "clientData1" to the specified value
- setClientData2(): set "clientData2" to the specified value
- setText(): set "text" to the specified value

Component and Inheritance Graphs: N/A

b.

Name: BasicComponent [You92]
Superclass: N/A
Aggregated classes: N/A

Description: Abstract class, superclass of all component classes. In their constructor, component classes take a widget as an argument that serves as the parent of the component's base widget "w" (root widget) and a character string as "name" of the object. BasicComponent provides methods to retrieve the base widget. This class is adopted from [You92].

Attributes:

- name: name of the instance of the object class
- w: the base widget of the instance

Methods:

- Constructor: protected constructor to prevent instantiation, takes the "name" as parameter.
- Destructor
- manage(): virtual function to manage the object.
- unmanage(): virtual function to unmanage the object.
- baseWidget(): returns the basewidget "w".

Component and Inheritance Graphs: N/A

c.

Name: CustomDialogCallData
Superclass: N/A
Aggregated classes: N/A

Description: Abstract object class. This is an auxiliary class to DialogContainer object class. It stores all relevant data to invoke external function(s) "f_o", "f_c", "f_h" instead of the default member functions, upon activating "ok", "Cancel", "Help" pushbuttons of DialogContainer.

Attributes:

- dialogContainer: pointer to an instance of DialogContainer object class.
- ok: pointer to a function which takes one parameter, (void *). This external function is the callback function of "ok" button of DialogContainer object class.
- cancel: pointer to a function which takes one parameter, (void *). This external function is the callback function of "cancel" button of DialogContainer object class.
- help: pointer to a function which takes one parameter, (void *). This external function is the callback function of "help" button of DialogContainer object class.
- clientData1, clientData2, clientData3: pointers to any data structure or object to facilitate the information exchange between object and external callback function.

Methods:

- Constructor: Takes all seven attributes to create an instance of the object class.
- Destructor
- return functions: A function is provided for each attributes to return them.
- set....(): functions are provided to set "clientData1", "clientData2", clientData3", "ok", "cancel", "help" to the specified value

Component and Inheritance Graphs: N/A

d.

Name: DialogCallbackData
Superclass: N/A
Aggregated classes: N/A

Description: Abstract object class. This is an auxiliary class to MessageDialog object class. It stores all relevant data to invoke external function(s) "f_o", "f_c", "f_h" instead of the default member functions, upon activating "ok", "Cancel", "Help" pushbuttons of MessageDialog.

Attributes:

- messageDialog: pointer to an instance of MessageDialog object class
- ok: pointer to a function which takes one parameter, (void *). This external function is the callback function of "ok" button of MessageDialog class.
- cancel: pointer to a function which takes one parameter, (void *). This external function is the callback function of "cancel" button of MessageDialog class.
- help: pointer to a function which takes one parameter, (void *). This external function is the callback function of "help" button of MessageDialog object class.
- clientData: pointer to any data structure or object to facilitate the information exchange between object and external callback function.

Methods:

- Constructor: Takes all five attributes to create an instance of the object class.
- Destructor
- return functions: A function is provided for each attributes to return them.
- set....(): functions are provided to set "clientData", "ok" "cancel", "help" to the specified value

Component and Inheritance Graphs: N/A

e.

Name: ListCallbackData
Superclass: N/A
Aggregated classes: N/A

Description: Abstract object class. This is an auxiliary class to List object class. It stores all relevant data to invoke external function "f" instead of the default member function upon the selection of item(s) in the list widget.

Attributes:

- list: pointer to an instance of List object class
- select: pointer to a function which takes one parameter, (void *) This external callback function is invoked upon the selection of item(s) in the list widget.
- clientData1, clientData2: pointers to any data structure or object to facilitate the information exchange between the object and external callback function.

Methods:

- Constructor: Takes all four attributes to create an instance of the object class.
- Destructor
- list(): returns "list"
- select(): returns "select"
- clientData1(): returns "clientData1"
- clientData2(): returns "clientData2"
- set....(): functions are provided to set "clientData1", "clientData2", "select" to the specified value

Component and Inheritance Graphs: N/A

f.

Name: MainMCallbackData
Superclass: N/A
Aggregated classes: N/A

Description: Abstract object class. This is an auxiliary class to MainMenu object class. It stores all relevant data to invoke external function "f" instead of the default member function upon activating any of MainMenu's pushbuttons.

Attributes:

- mainMenu: pointer to an instance of MainMenu object class
- push: pointer to a function which takes one parameter, (void *). This external callback function is invoked upon pressing a pushbutton that belongs to the relevant instance of MainMenu object class.
- clientData0, clientData1, clientData2: pointers to any data structure or object to facilitate the information exchange between object and external callback function.

Methods:

- Constructor: Takes attributes "mainMenu", "clientData1", "clientData2", "push" to create an instance of the object class.
- Destructor
- mainMenu(): returns "mainMenu"
- push(): returns "push"
- clientData0(): returns "clientData0"
- clientData1(): returns "clientData1"
- clientData2(): returns "clientData2"
- set....(): functions are provided to set "clientdata0", "clientData1", "clientData2", "push" to the specified value

Component and Inheritance Graphs: N/A

g.

Name: MessageManager
Superclass: N/A
Aggregated classes: N/A

Description: Abstract object class. The basic task of MessageManager is to return a MessageDialog of the appropriate type, displaying a predefined message on demand. All existing MessageDialogs are created by a MessageManager object. The parent of the MessageManager is the parent of all MessageDialogs it is responsible for. It keeps track of MessageDialogs it has created and destroys them if requested. MessageManager reuses the free dialogs.

Attributes:

- parent: Parent of the instance of MessageManager object class.
- name: Name of the instance of MessageManager object class.
- messageList: Initially empty list of MessageDialog's.

Methods:

- Constructor: "parent" and "name" have to be given to construct an object.
- Destructor
- getDialog(): Gets an unused dialog from "messageList" or create a new one and returns it.
- createMessage(): Virtual function to create a MessageDialog object. It is placed at the front of "messageList" upon creation.
- post(): Locate a MessageDialog object using "getDialog()" and "createMessage()" functions. Type, message, buttons and callback functions are given as parameters to that function.
- cleanUp(): Marks all MessageDialogs "unused" in "messageList".
- emptyMessageList(): Free all nodes in "messageList".
- getname(): Returns name of MessageManager object.
- getparent(): Returns "parent" widget
- getList(): Returns "messageList"

Component and Inheritance Graphs: N/A

h.

Name: PromptCallbackData
Superclass: N/A
Aggregated classes: N/A

Description: Abstract object class. This is an auxiliary class to PromptDialog object class. It stores all relevant data to invoke external function(s) "f_o","f_c","f_h","f_t" instead of the default member functions, upon activating "ok", "Cancel", "Help" pushbuttons and pressing enter in text-field.

Attributes:

- promptDialog: pointer to an instance of PromptDialog object class
- text: pointer to a function which takes one parameter, (void *) This external function is the callback function if return has been pressed in the prompt text.
- ok. pointer to a function which takes one parameter, (void *). This external function is the callback function of "ok" button of PromptDialog class.
- cancel: pointer to a function which takes one parameter, (void *). This external function is the callback function of "cancel" button of PromptDialog class.
- help: pointer to a function which takes one parameter, (void *). This external function is the callback function of "help" button of PromptDialog object class.
- clientData1, clientData2: pointer to any data structure or object to facilitate the information exchange between the object and external callback function.

Methods:

- Constructor: Takes all seven attributes to create an instance of the object class.
- Destructor
- return functions: A function is provided for each attributes to return them.
- set....(): functions are provided to set "clientData1", "clientData2", "text" to the specified value

Component and Inheritance Graphs: N/A

i.

Name: RadioTCallbackData
Superclass: N/A
Aggregated classes: N/A

Description: Abstract object class. This is an auxiliary class to RadioToggle object class. It stores all relevant data to invoke external function "f" instead of the default member function, upon activating any of RadioToggle's toggle-buttons.

Attributes:

- radioToggle: pointer to an instance of RadioToggle object class
- toggle: pointer to a function which takes one parameter, (void *) . This external callback function is invoked upon pressing a toggle-button that belongs to the relevant instance of RadioToggle object class.
- clientData1, clientData2: pointers to any data structure or object to facilitate the information exchange between the object and external callback function.

Methods:

- Constructor: Takes the four attributes to create an instance of the object class.
- Destructor
- radioToggle(): returns "radioToggle"
- toggle(): returns "toggle"
- clientData1(): returns "clientData1"
- clientData2(): returns "clientData2"
- set....(): functions are provided to set , "clientData1", "clientData2", "toggle" to the specified value

Component and Inheritance Graphs: N/A

j.

Name: TableFCallbackData
Superclass: N/A
Aggregated classes: N/A

Description: Abstract object class. This is an auxiliary class to TableFill object class. It stores all relevant data to invoke external function "f" instead of the default member function upon pressing Enter in any of TableFill's text-fields.

Attributes:

- tableFill: pointer to an instance of TableFill object class
- text: pointer to a function which takes one parameter, (void *). This external callback function is invoked upon pressing enter in one of the text fields of the relevant instance of TableFill object class.
- clientData1, clientData2: pointers to any data structure or object to facilitate the information exchange between object and external callback function.

Methods:

- Constructor: Takes the four attributes to create an instance of the object class.
- Destructor
- tablefill(): returns "tableFill"
- text(): returns "text"
- clientData1(): returns "clientData1"
- clientData2(): returns "clientData2"
- set....(): functions are provided to set , "clientData1", "clientData2", "text" to the specified value

Component and Inheritance Graphs: N/A

k.

Name: UIComponent [You92]
Superclass: BasicComponent
Aggregated classes: N/A

Description: Abstract class, derived from BasicComponent. Supports the creation of new derived classes through built in error handling (e.g.: assert() function). This class is adopted from [You92].

Attributes: None

Methods:

- Constructor: Protected constructor to prevent instantiation, takes "name" as parameter.
- Destructor
- installDestroyHandler(): Installs widgetDestroy() function. This function is needed because the base widget of the object, "w" is specified only after object creation.
- widgetDestroyed(): Assigns NULL "w" to avoid referencing destroyed objects.
- manage(): Manages the base widget of the object.

Component and Inheritance Graphs: N/A

Chapter 4

Designing the GUI for Course Advisor System

All the research on interface design principles focuses around two major standpoints that can be said with the phrase "know the user, know the task" [Han71]. If the developer knows the user and the task, he/she can choose between the two ways the design can be conceptually carried out: The user is adapted to the system or the system is adapted to the user[Ian92]. Developers aim for the latter approach since the former would result in intensive user training and the users would loose their natural behavior while interacting with the system. First we describe the potential users of CAS and the interaction styles the user interface should have, to cope successfully with the needs of the users. Following that we describe the realized User Interface for CAS.

4.1 Users and Task of CAS

Among the several groupings of users the one presented in [Shn87] is applied in this case. Users are classified into three groups:

- **Novice users** have no syntactic knowledge[1] and little semantic knowledge[2].
- **Knowledgeable intermittent users** may have problems with syntactic knowledge but they possess good semantic knowledge of the task and general computer concepts as well.

[1]knowledge of device dependent low-level details (CTRL-C, ESCAPE keys etc.)

[2]consists of knowledge of computer concepts and task concepts

- **Frequent users** have good syntactic and semantic knowledge, they aspire to complete their work quickly.

The intended users of CAS are (as described in Section 1.5.2) undergraduate Computer Science students seeking advice prior to registration. Accordingly it can be assumed that the potential users of CAS have knowledge and practice with interfaces of various software packages and are aware of the task (advising and registration process). Consequently users of CAS are classified to be **Knowledgeable intermittent users** who "maintain semantic knowledge of the task and computer concepts" [Shn87] but due to their casual use of CAS they may not have good syntactic knowledge. Students registering the first time obviously have less semantic knowledge of the task (advising procedure) as students registering in the third year. To encourage students to use CAS, the developed interface can be used on different levels of complexity:

- The student identifies himself/herself by entering his/her ID and instructs CAS to suggest courses.
- The student identifies himself/herself by entering his/her ID, enters the courses he/she prefers to take and some other constraints. Moreover the student can differentiate between constraints (Soft-Constraints) that can be changed through the iterations of advising and constraints (Hard-Constraints) that can not be relaxed during an advising session[3]
- Student can take a more active role in communication with the system asking explanation, modifying Soft-Constraints, initiating new iterations, rejecting or replacing some of the suggested courses etc.

This way the users can accomplish their task and can get feasible output from the system even with little or no syntactic knowledge, and can advance to a more complex level without training.

The task of CAS has been described in detail and analyzed in Chapter 2. The interaction style of the user interface of CAS is menu driven. The user navigates by means of various buttons and menus. We used RML to implement the user interface. In the relevant literature it is pointed out that the most important principle of UI design is consistency. OOD promotes consistency, since it aims to create reusable

[3]from entering student ID till the acceptance of the suggested courses or cancelation

components. In UI design creating a set of object classes, and using them systematically as building blocks of the interface will contribute to the consistency of UI. One can say that Motif toolkit defines a "look and feel" of the interface and RML enhances consistency through its larger granularity of objects it supports.

4.2 Implementation of the User Interface of CAS

The user interface of CAS consists of the following modules:

- Control function: Initializes CAS (the application), constructs and displays the GUI component.
- GUI component: Detailed description is given in the following section.
- Message pool: Stores all possible messages the system can display to the user.
- External functions: A set of external functions that serve as callback functions.

The user interface communicates with CAS through a Communication layer that is outlined in Section 4.2.2.

4.2.1 Description of the GUI component

The graphical components of GUI are gathered into an object class named CasFace. Only one instance of CasFace is created while running the application, by the Control function (main function). From the viewpoint of the user, CasFace is the main application window. The suggested structure of the Main Window follows the suggestion given in *Motif Style Guide* [Fou93]. In a typical Main Window there are a menu-bar, a work-area, a command area and a message area arranged one below the other vertically. The work area is holding the main interface object, like Text widget in the case of an editor, or a drawing area etc. The command area is to enter commands (user to the system), the message area displays messages to the user (system to the user). We have structured the main application window keeping in mind these recommendations. In the following subsections we describe CasFace object class through its visualization and functionality. Its Component Graph (Figure 48) shows clearly how CasFace has been aggregated from the object classes of RML, and Figure 49. demonstrates how the GUI passes through the different states during an advising session. There is one issue to be discussed before going on.

Object creation and destruction policy

Objects can be created and destroyed any time during program execution. However it is wise to set out a policy. For instance all objects can be created at the beginning of the program, and destroyed at the end. With this method, keeping track of the objects is easy, and fast. However, initialization of the application is longer and the necessity of all objects must be known at that time. If the objects are dynamically created during the running of the application, care has to be taken to record the objects (retain a pointer to it) to later reference. CasFace is built by using a mixture of the two policy. The objects that are essential, are created by the constructor of CasFace. They are active components of CasFace and they are destroyed only together with CasFace object. The objects with unpredictable existence are created when needed. To keep track of these objects created during the lifetime of the application, special manager classes should be created. These managers control the creation destruction, and reuse of the objects.

Main Window of the application

A PanedWindow widget was chosen as manager of the main application window. "The Paned widget lays out its children in a vertically-tiled format" [HF94]. The panes are separated by control sashes that can be used for resizing the panes. We used the sashes through the development and set them invisible when the desired look of the UI had been achieved. Three areas are set up in the application window, main menu, work and message areas laid out from top to bottom. See Figure 38. In the main menu area there are three pushbuttons ("CANCEL", "QUIT", "HELP") and two toggle buttons ("Hard User-Constraints", "Explanation on Preferred Courses"). These buttons are permanently visible on the screen, but their sensitivity is changed from time to time to make the user's navigation easier. We did not use menu-bar here because there are only few items, and conditions for a possible grouping of these items are diverse. The work area is where the user's attention is focused through an advising session. Windows are opened, and dialogs popped up in the work area. If CAS is idle, the interface prompts for the ID of a student in the work area. The message area is a scroll-able text.

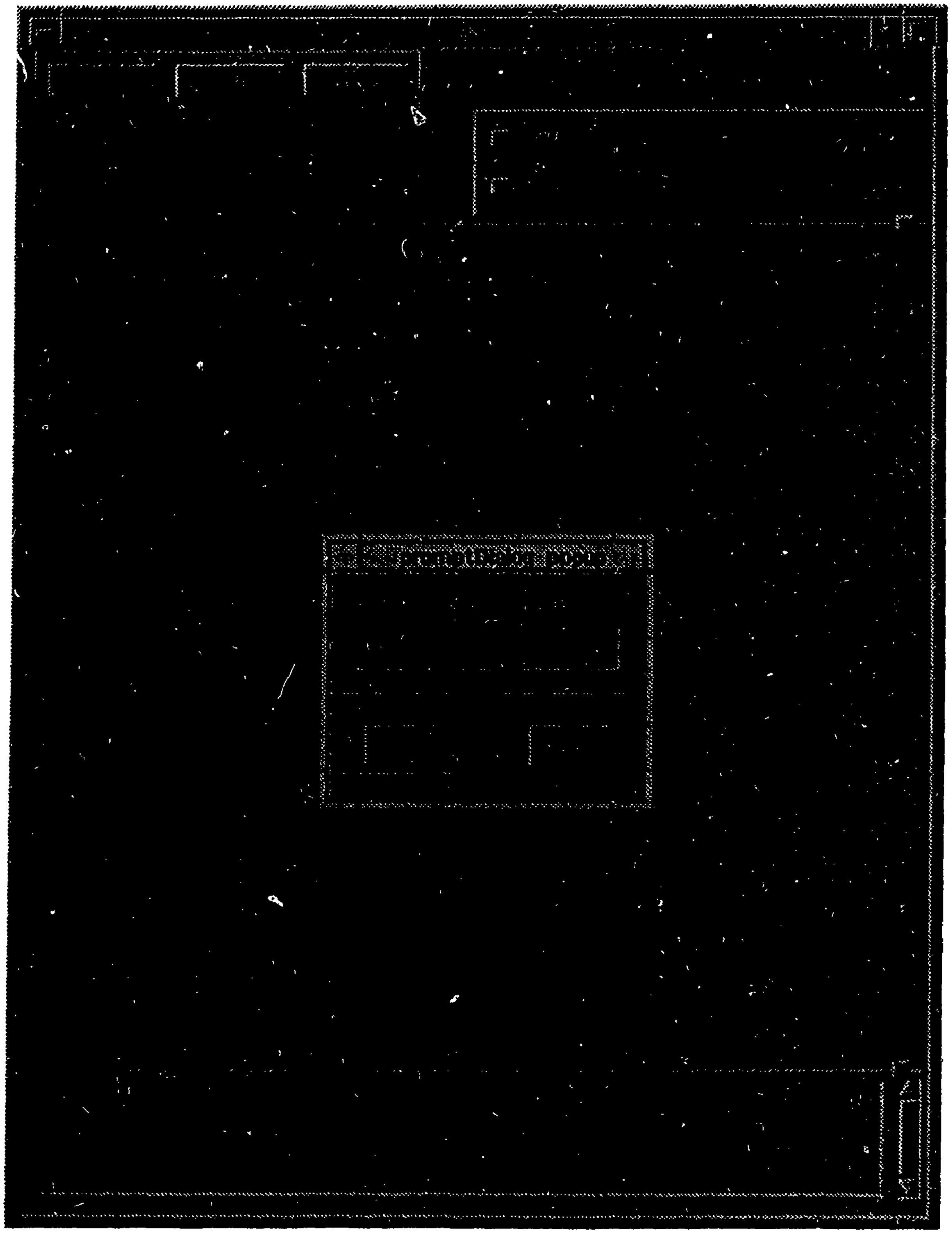


Figure 38: Main Application Window of CAS

The three pushbuttons are implemented using a **MainMenu object**. Activating "CANCEL" will cancel the advising session meaning to clear all information entered or implied (ID number, various constraints, preferred courses, suggested courses, message area), destroy dispensable objects (message dialogs) and display the initial Main application window (prompt for student ID and set the sensitivity of main menu area). CAS is informed of cancelling the advising session through "ResetAdvise()" function of the Communication layer (See 4.2.2). In that state of the user interface "CANCEL" is insensitive since this is the initial state of the system. Activating "QUIT" will exit the application "HELP" can provide on line help. The two toggle buttons are implemented using a **RadioToggle object**, they are insensitive in this state. Their function will be explained in the relevant section below. The message area contains a **MessageArea object**. This is a non edit-able scroll-able Text widget. All the messages occuring during an advising process are shown there. The most recently displayed message is highlighted by default. Naturally it is possible to display only the most recent message, but we thought it is advantageous to be able to review a particular scenario of messages. Significant messages and questions are displayed by **MessageDialog objects.** A **MessageManager object** is responsible for all MessageDialog objects (as it has been discussed above). A **PromptDialog object** is displayed in the work area of the main window. The student can enter his/her ID number. Upon pressing "Return" or "OK" button of the dialog, ID will be sent to CAS through the Communication layer for verification. See "ValidateID()" function in section 4.2.2. If invalid ID number has been entered, a message will be displayed in the message area and the PromptDialog will be cleared. Otherwise the next state of the interface (Preferred courses and Constraints Entry) will be displayed.

Preferred courses and Constraints Entry

In this state all buttons of the main menu area are sensitive. A menu-bar with three menu-title appears on the top of the working area. Each menu-title has a pull-down menu. The titles from left to right are "Advise", "Options" and "Help". The menu-bar is implemented with a **ContraintMenu object**. The user enters commands for CAS through "Advise" menu-title, and provides information to facilitate the task of CAS through "Options". "Help" provides the hook for on-line help facilities.

Three menu-items belong to the pull-down menu of "Advise" title, each of them invokes a callback function to access the Communication layer:

Save Options: Saves the information entered in "Options". The information entered is saved in a global User Constraint object. "User Constraint" object class is implemented by the application, CAS. CAS maintains an instance of User Constraints class in the Communication layer (See 4.2.2) and permits write access for UI. This object is named "Soft User-Constraints".

Proceed...: Instructs CAS through the Communication layer to give advice with respect to the available information. See "StartAdvise()" function in section 4.2.2. The user interface will enter to "Suggesting Courses" state (described below).

Stop Process: Instructs CAS through the Communication layer to stop suggesting and discard the already selected courses. See "StopAdvise()" function in section 4.2.2. However all the information entered by the user or obtained by CAS will be retained and reused when "Proceed..." is reactivated.

The pull-down menu of "Options" consists of five menu-items There is an object associated with each menu-item. These objects are customized dialog boxes. Clicking on the menu-item, pops up the dialog box and the user can enter, modify or clear the data:

Preferred Courses...: Student can enter the courses he/she prefers to take. The course can be fully specified with the string "Course ID/Section" (COPM291/XX) or if the student does not know which section would be suitable, "Course ID" (COMP291). Associated object is **ListDialog object**. See Figure 39.

Unpreferred Courses...: Student can enter the courses he/she does not want to take. If the course is specified with the string "Course ID/Section" (COPM291/XX) it means the student does not want to take that particular Section, but might not reject other sections of the same course. This task is implemented with a **ListDialog object**. See Figure 39.

Inconvenient time...: Student can specify the time periods from Monday to Friday when he/she can not or does not want to attend classes. This functionality is implemented with a **FromList object**. See Figure 40.

Campus...: Student can specify if he/she can not or does not want to attend classes held on a particular campus. Implemented with a **ToggleDialog object**. See Figure 40.

Workload...: Student can specify the minimum and maximum number of credits, minimum and maximum number of courses he/she wants to take. Implemented with a **TableFillDialog object**. See Figure 10.

This menu makes it possible for the user to enter constraints for CAS to take into consideration. These constraints can be modified, during the advising process. For example the user can decide to change the "Inconvenient time" constraint by allowing taking courses on Friday, even though at first he/she marked it as inconvenient. The new constraints have to be saved by "Save Options" and in the next iteration (pressing "Proceed...") CAS is going to consider them. These constraints are called "Soft User-Constraints" and the menu will be referred to as "Soft Constraints Menu" through this work. It is important to note that entering any constraints is not mandatory. The user can simply give the command to "Proceed..." and CAS will suggest what courses to take for the given student's transcript, and the course schedule.

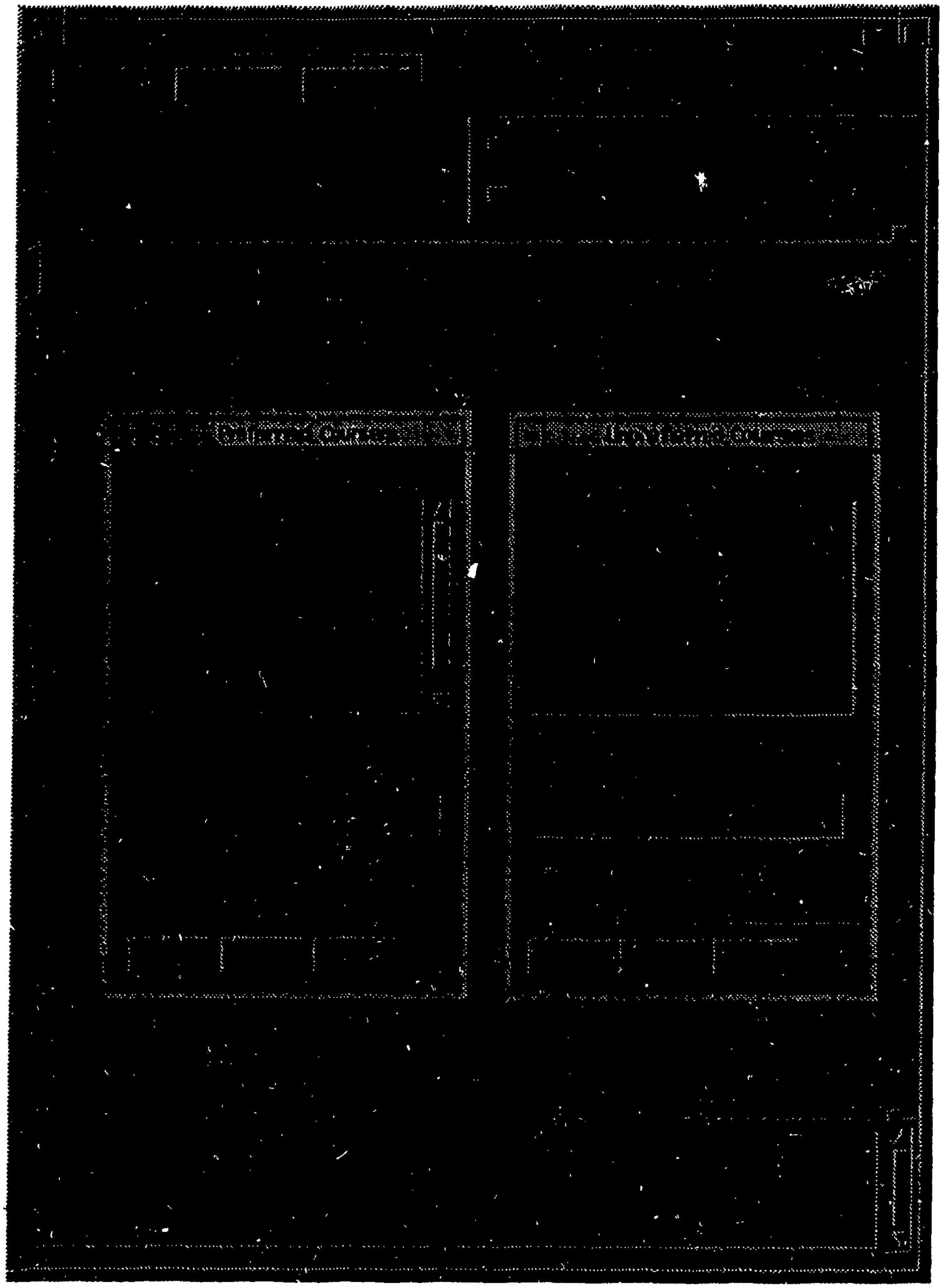


Figure 39: User constraints - part 1

√

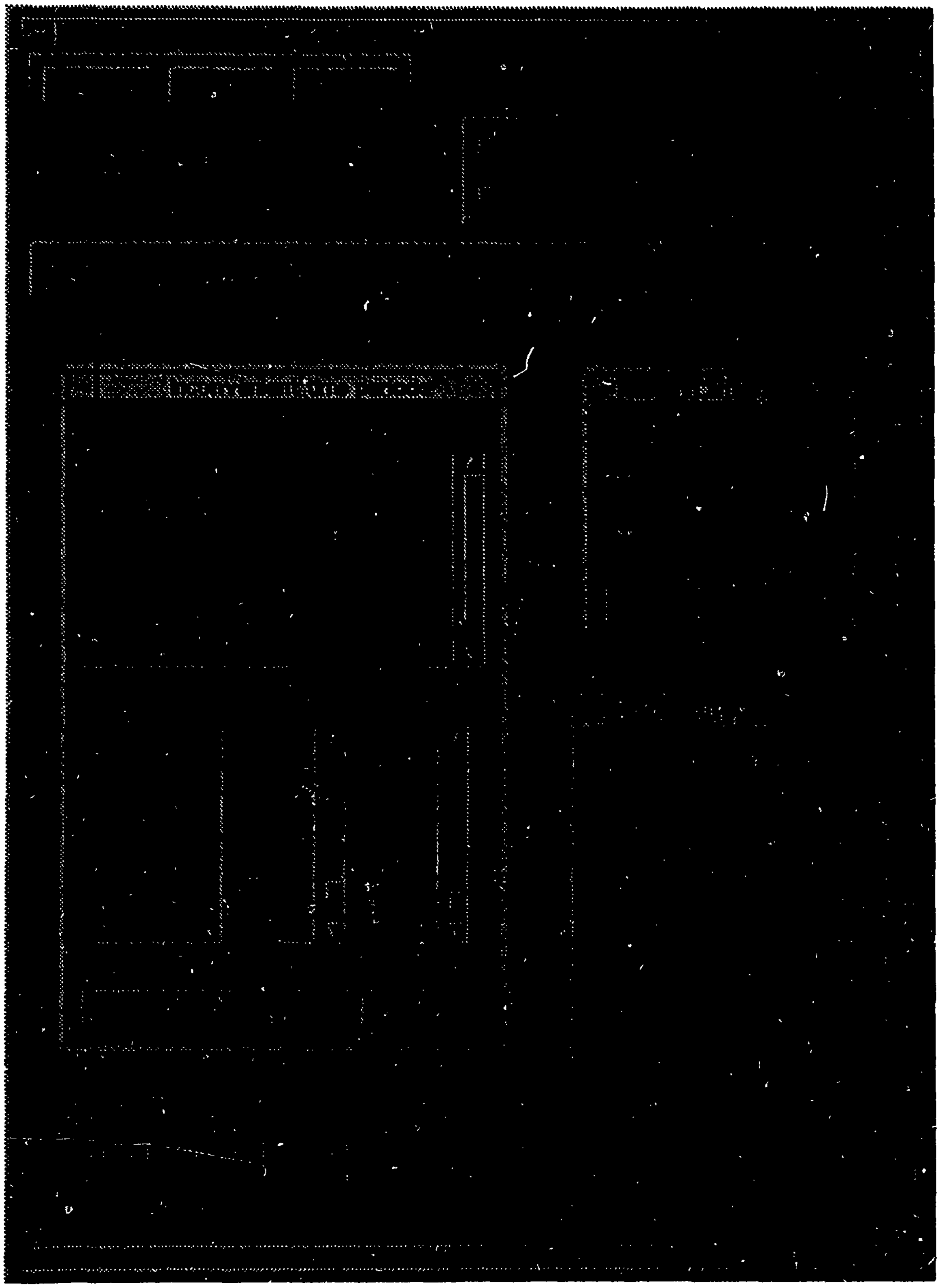


Figure 40: User constraints - part 2

Hard User-Constraints Entry

Pressing "Hard User-Constraints" toggle-button in main menu area will pop up a window in the work area. The user can enter "Hard User-Constraints" here. Contrary to "Soft User-Constraints", "Hard User-Constraints" can be saved only once during an advising session. A MessageDialog reminds the user, upon popping the window up, about this aspect. The message dialog is controlled by the MessageManager object. In the top of the window there is a menu-bar realized by a **ConstraintMenu object**. There are three menu-titles on the menu-bar, each has a pull-down menu. The titles from left to right are "Proceed", "Options" and "Help". See Figure 41. The user enters commands through "Proceed" menu-title, and not relax-able constraints through "Options". "Help" provides the hook for on-line help facilities.

Three menu-items belong to the pull-down menu of "Proceed" title, each of them invokes a callback function:

Save Options: Saves the information entered in "Options". The information entered is saved in a global User Constraint object (See 4.2.2). As in the case of "Soft User-Constraints" object, CAS maintains an instance of User Constraints class in the Communication layer for the storage of data entered in this window. This object is named "Hard User-Constraints".

View Options: Initially this menu-item is insensitive. Upon saving the options (selecting "Save Options" in this pull-down menu) its sensitivity will be set True. There is a **BoardListDialog object** associated with this menu-item. Clicking on "View Options" will pop up a dialog box where the user can view the already saved constraints (Figure 42).

Quit: Pops down the window.

The pull-down menu of "Options" menu-title consists of three menu-items, "Unpreferred courses...", "Inconvenient Time..." and "Campus...". Each has an object associated with it. Their role and implementation are similar to the corresponding objects in "Soft Constraints Menu". We introduced "Hard User-Constraints" to make the process of selecting the courses faster. For example, if a student works on Monday and Thursday from 1:00pm to 7:00pm, the courses offered within this interval are definitely not suitable, so CAS should not waste time by looking at them.

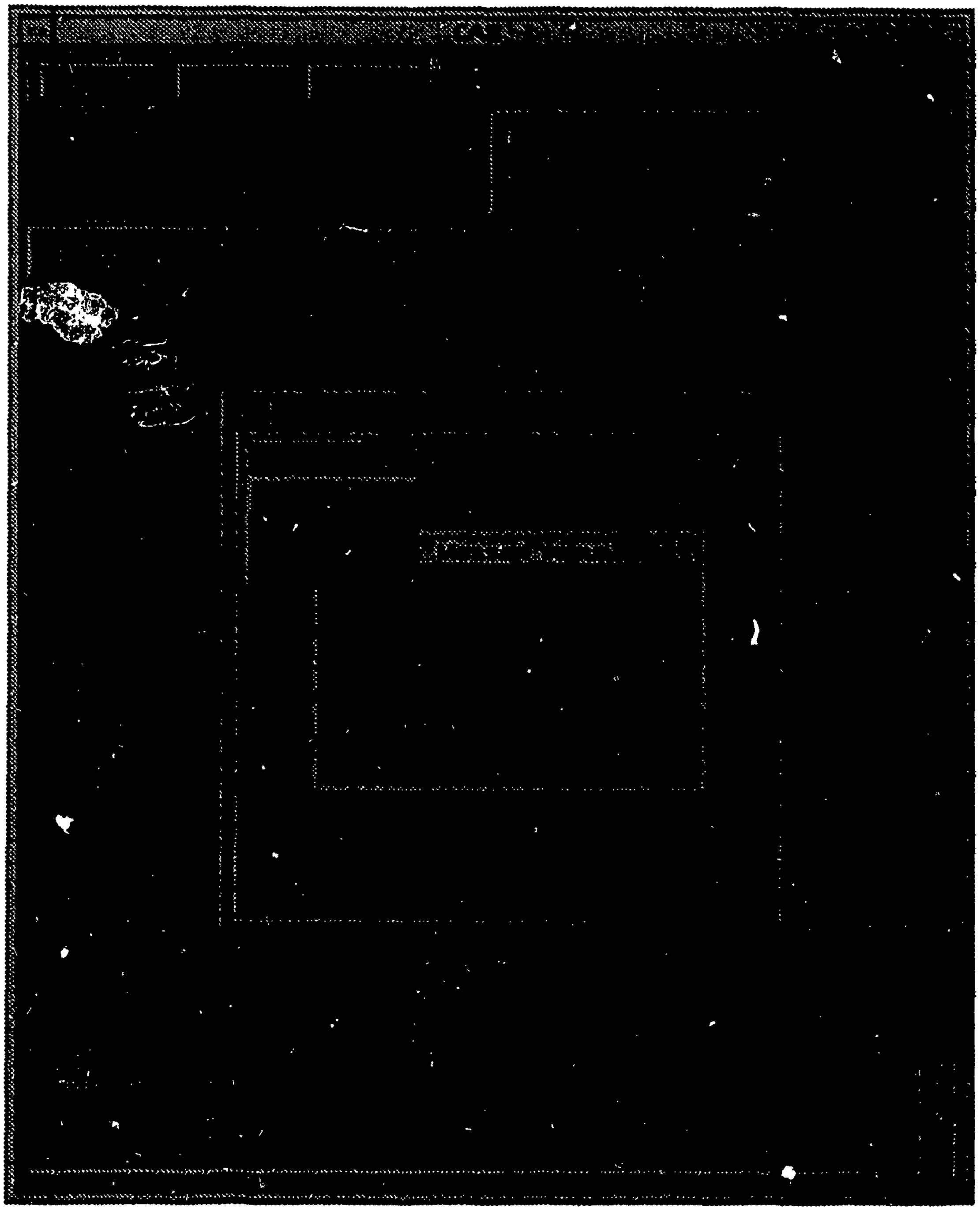


Figure 41: Hard-Constraints Menu

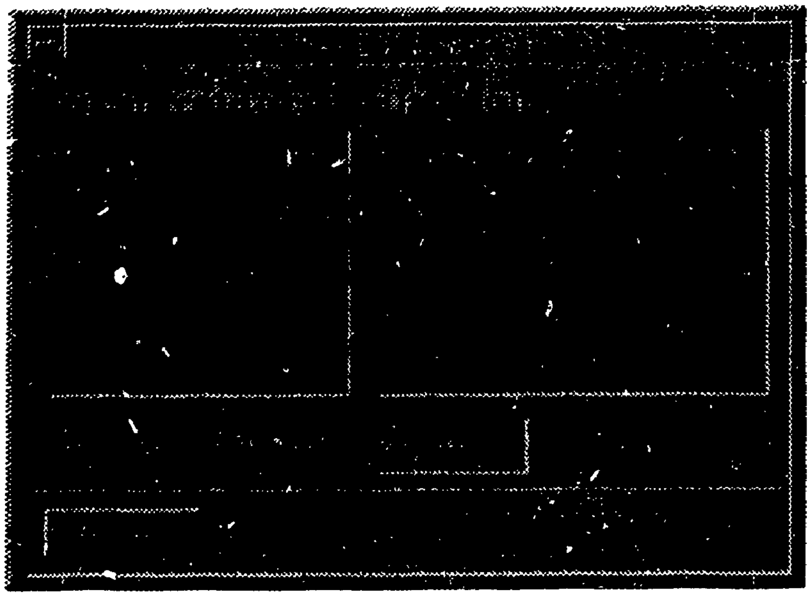


Figure 42: View Hard Constraints

Suggesting Courses

Selecting "Proceed..." menu-item from "Soft Constraints Menu" invokes "StartAdvise()" function of the Communication layer and move UI to Suggesting Courses state. A window appears in the work area containing an **ActiveEditTable object**: a table, underneath in a border a text field and a list. There is a row of buttons at the bottom of the window with labels "OK", "MODIFY", "DELETE", "CHANGE", "CANCEL", "REJECT", "HELP". The return value of "StartAdvise()" indicates that CAS is ready with its suggested course list, and it is available in the global SuggestedCourses object (accessible for UI through the Communication layer, see 4.2.2). The suggested courses are displayed in the table. "DELETE", "CHANGE" and "CANCEL" buttons are insensitive. See Figure 43. The student can accept the entire list of suggested courses, by pressing "OK" button (this action will put the interface into "Confirmation" state) or reject all the courses by pressing "REJECT". The interface appends the rejected courses to the "Unpreferred Courses" constraint of "Soft Constraint Menu". If the student wants to keep some courses, he/she can modify the suggested list by pressing "MODIFY" pushbutton. The actions taken upon clicking on "MODIFY" are described in Modification subsection.

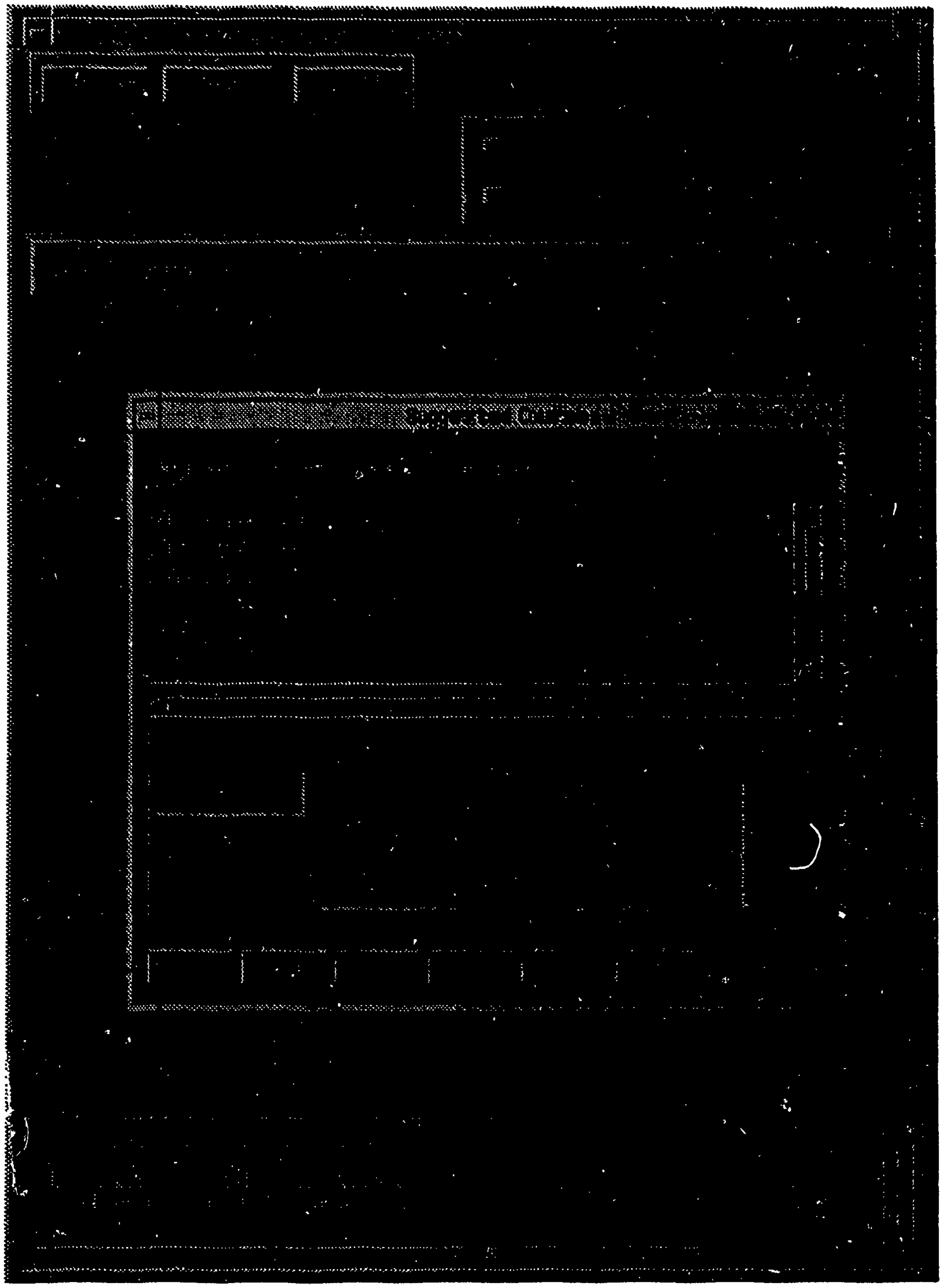


Figure 43: Suggesting Courses

Confirmation

The system enters into "Confirmation" state upon pressing "OK" button in "Suggesting Courses" state. A question type **MessageDialog object** pops-up posting the question "Do you want more courses to be selected for you?". See Figure 44. "YES" means the student wants to take more courses but keep the already suggested (possible modified) list of courses. This is accomplished by calling the "Suggest-MoreCourse()" function of the Communication layer. That function instructs CAS to suggest more courses while keeping the already accepted suggested courses. "NO" means the already suggested (and possibly modified) course list is sufficient enough. Pressing "NO" button takes the interface to "Display output" state.

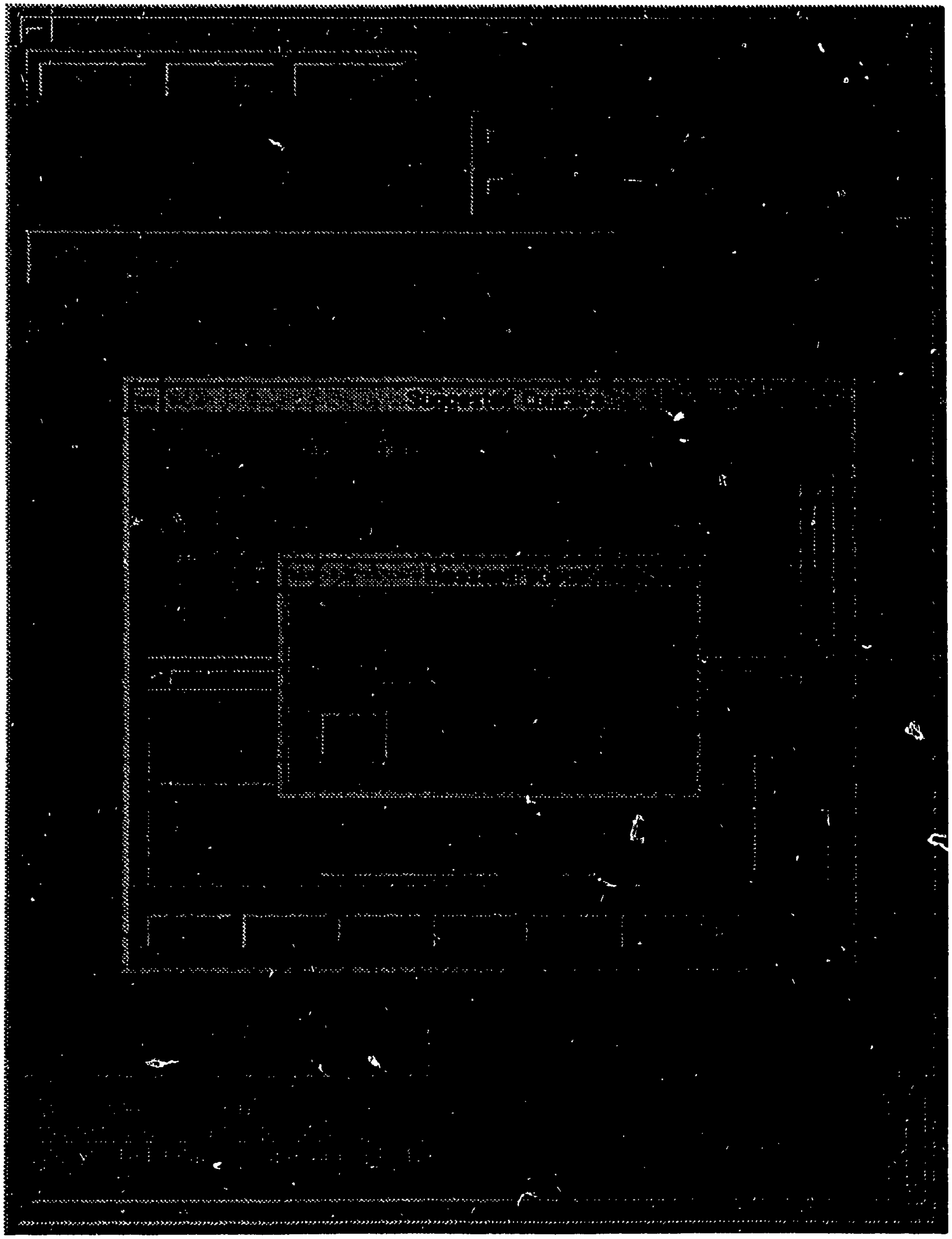


Figure 44: Confirmation of Suggested Course List

Display output

This state is the final state of the interface. The confirmed, suggested course list is displayed in a table. The table is contained by a **TableDialog object** that pops-up if the student has the desired number of confirmed courses. See Figure 45. The suggested list can be printed (written in file "suggestion.out") pressing "PRINT" , or acknowledge pressing "OK" button in the dialog box. After clicking on any of the two buttons, the dialog box pops-down and the system returns to its idle, initial state and prompts for the ID number of the next user. The interface indicates the end of advising session to CAS through "ResetAdvise()" function of the Communication layer (See 4.2.2).

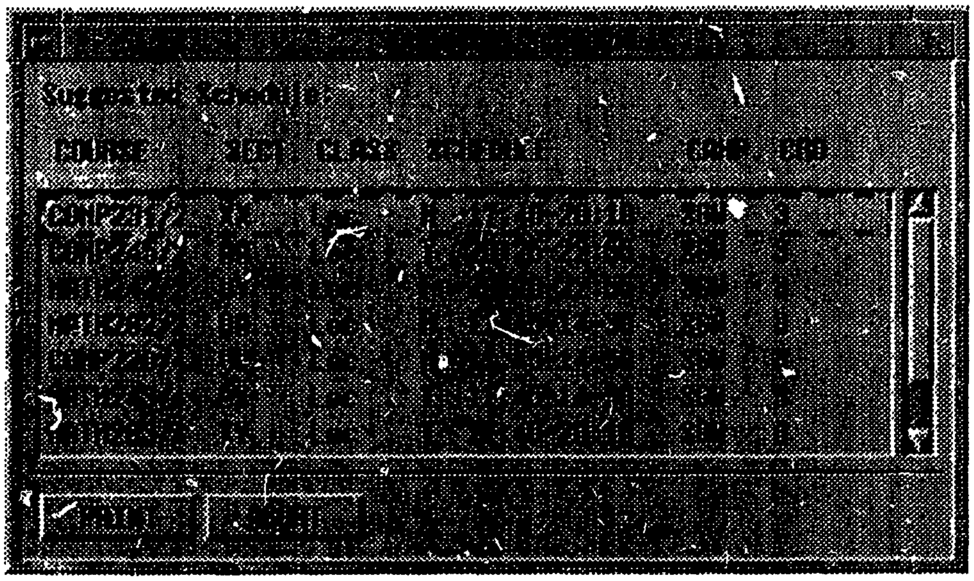


Figure 45: Suggested Course List (Output of CAS)

Modification

Modifications can be performed to the "Suggested Courses" window following the activation of "MODIFY" button. "MODIFY" and "REJECT" buttons become insensitive in this state. There are four buttons to manipulate the items of the table:

OK: Activating "OK" button means all wanted changes are made and the interface returns to the previous (Suggesting Courses) state.

DELETE: The student selects the course he/she wants to delete from the list. "DELETE" invokes "DelSuggestCourse()" function of the Communicate layer that asks permission from CAS for deletion. The selected course can be deleted only if CAS allows it. For instance if CAS does not allow to delete course 'A' (because course 'A' is co-requisite of course 'B' which is also in the suggested list) the student can delete course 'B' and after course 'A', or can decide to return to "Suggesting Courses" state, reject all the courses (possibly enter new user constraints) and initiates new iteration by pressing "Proceed..." in "Soft Constraints Menu".

CHANGE: Replaces a course with the same course with different schedule. (different section of the same course). The scenario of replacing a course is the following. The student selects the course he/she wants to replace. Presses "CHANGE" button. The interface asks the schedule of available sections of the selected course through "RequestCourseSched()" function of the Communication layer. CAS returns the schedule and it will be displayed in the table located in the lower half of **ActiveEditTable** object in "Suggested Courses" window. See Figure 46. The text field holds the course prefix and number and the table. The student selects the desired schedule, and presses "CHANGE". The user interface asks CAS (through "ChangeSuggestCourse()" function of Communication layer) to carry out the requested change. CAS agrees or denies to change its suggested list. (CAS will deny the request if it conflicts with the schedule of other suggested courses.) Positive response results, the schedule of the course in the suggested course list to be replaced by the selected schedule.

CANCEL: Discards all changes and returns to "Suggesting Courses" state.

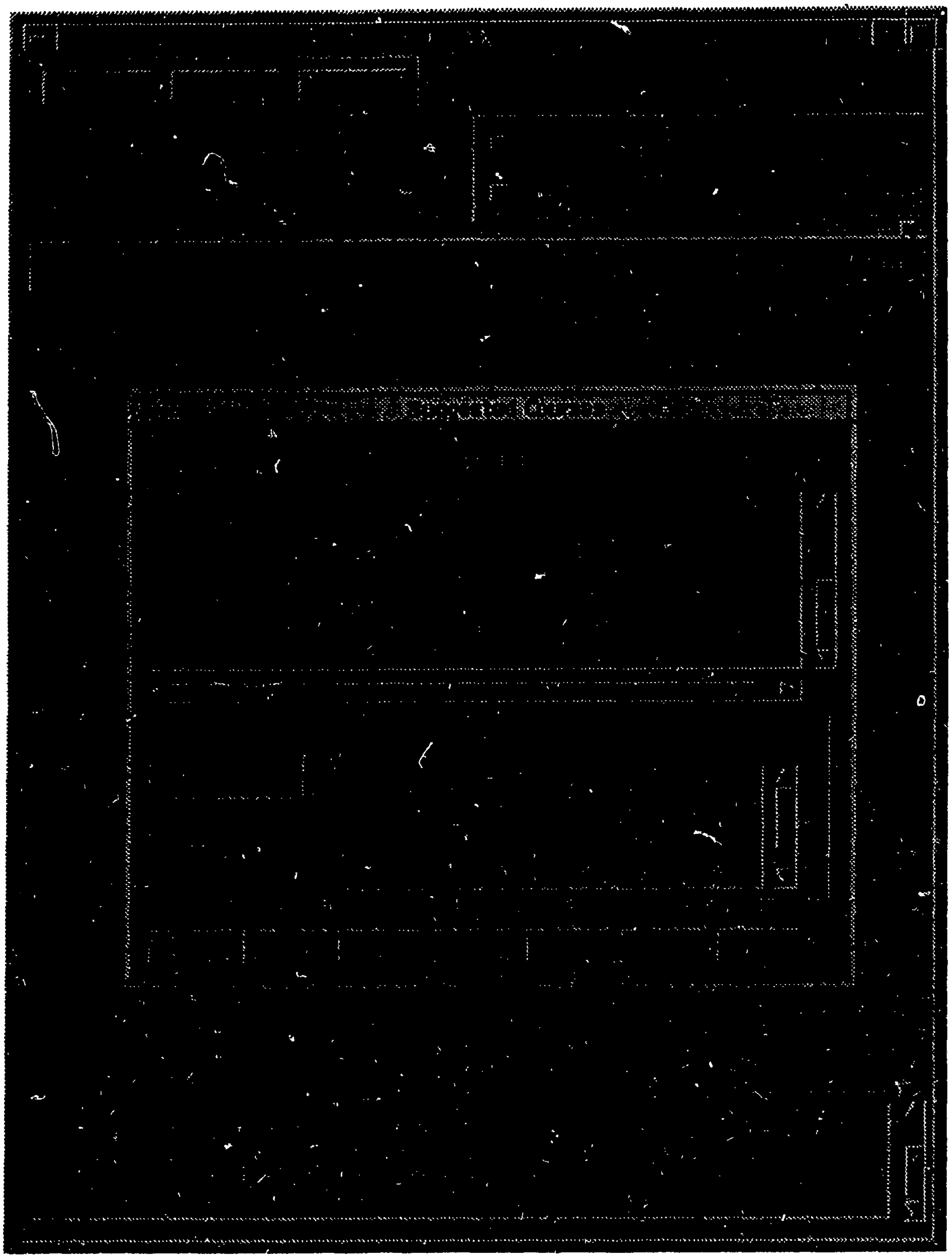


Figure 46: Changing the section of a suggested course

Explanation on Preferred courses

CAS provides explanation on preferred courses that is displayed by pressing the toggle-button (with "Explanation on Preferred courses" label) in main menu area. The preferred courses are entered in the "Soft Constraints Menu". CAS decides if they can or can not be chosen. If a course is not selected, CAS gives the reason for rejection: pre/co-requisite, availability, collision with other preferred course, qualification for the degree and work overload (too many preferred courses). The facility is implemented using **TableDialog object**. See Figure 47. Pressing the toggle-button pops-up the dialog box and activating "DISMISS" (the only pushbutton of the dialog box) pops-it down. Explanation on Preferred courses can be called any time and it does not alter the state of the system.

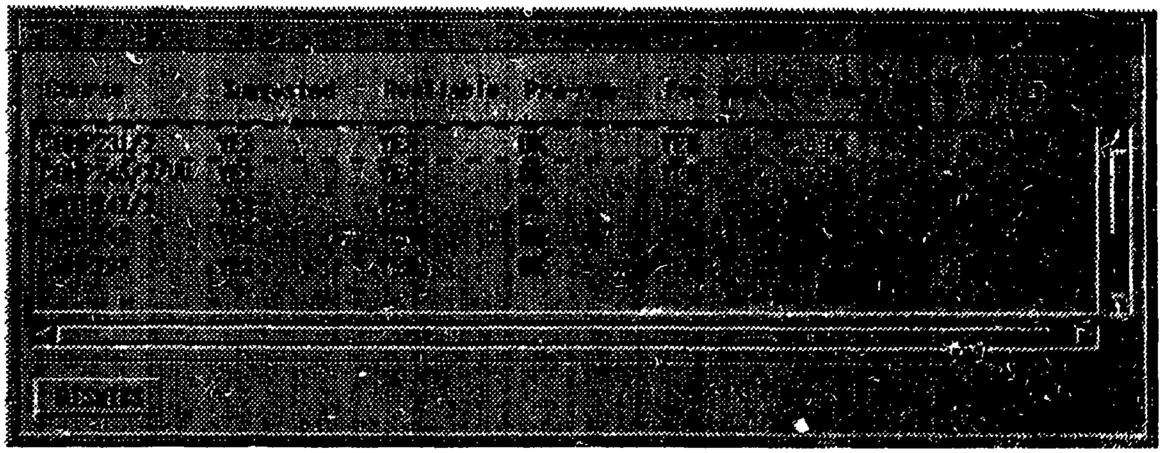


Figure 47: Explanation facility

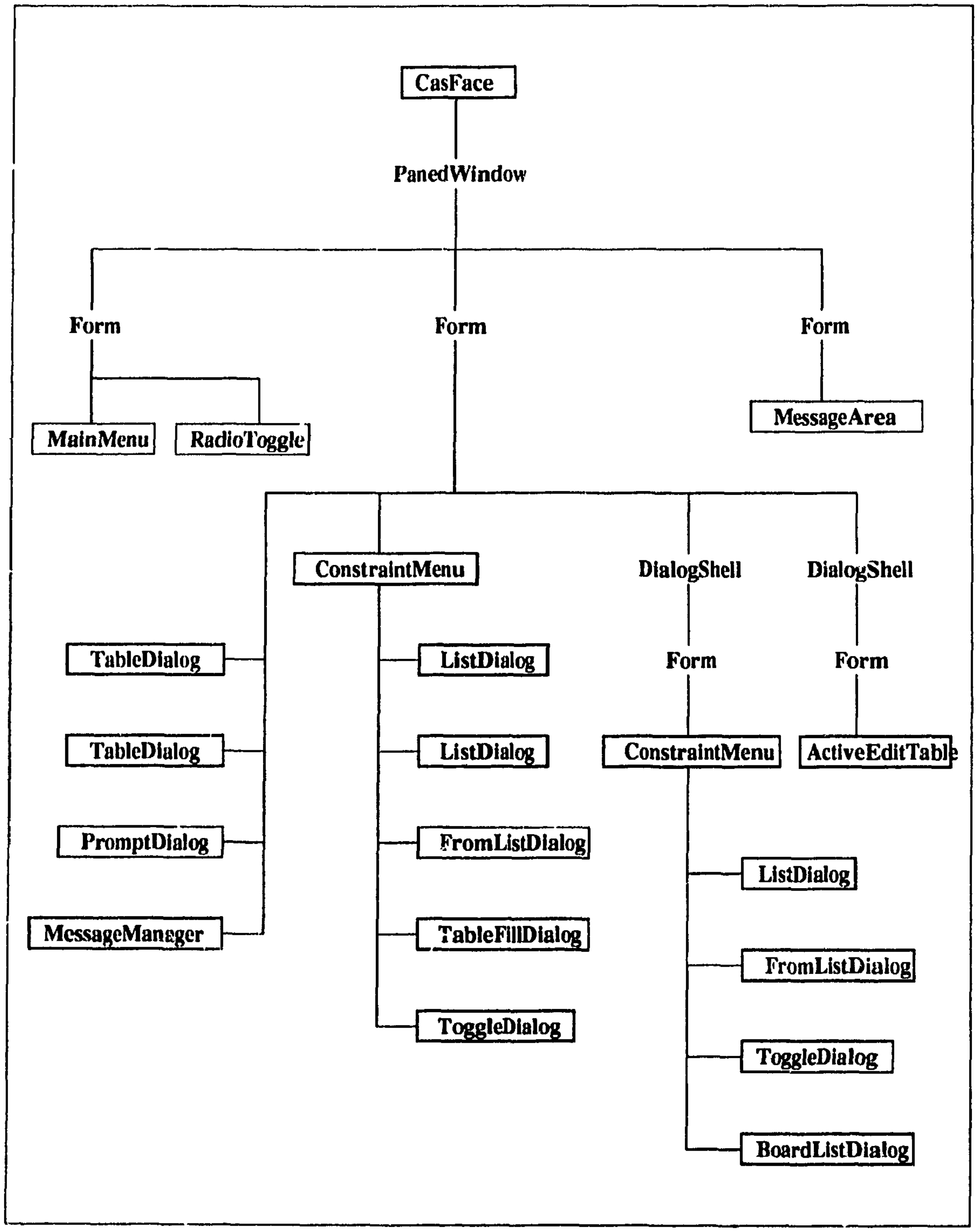


Figure 48: Component Graph of CasFace object class

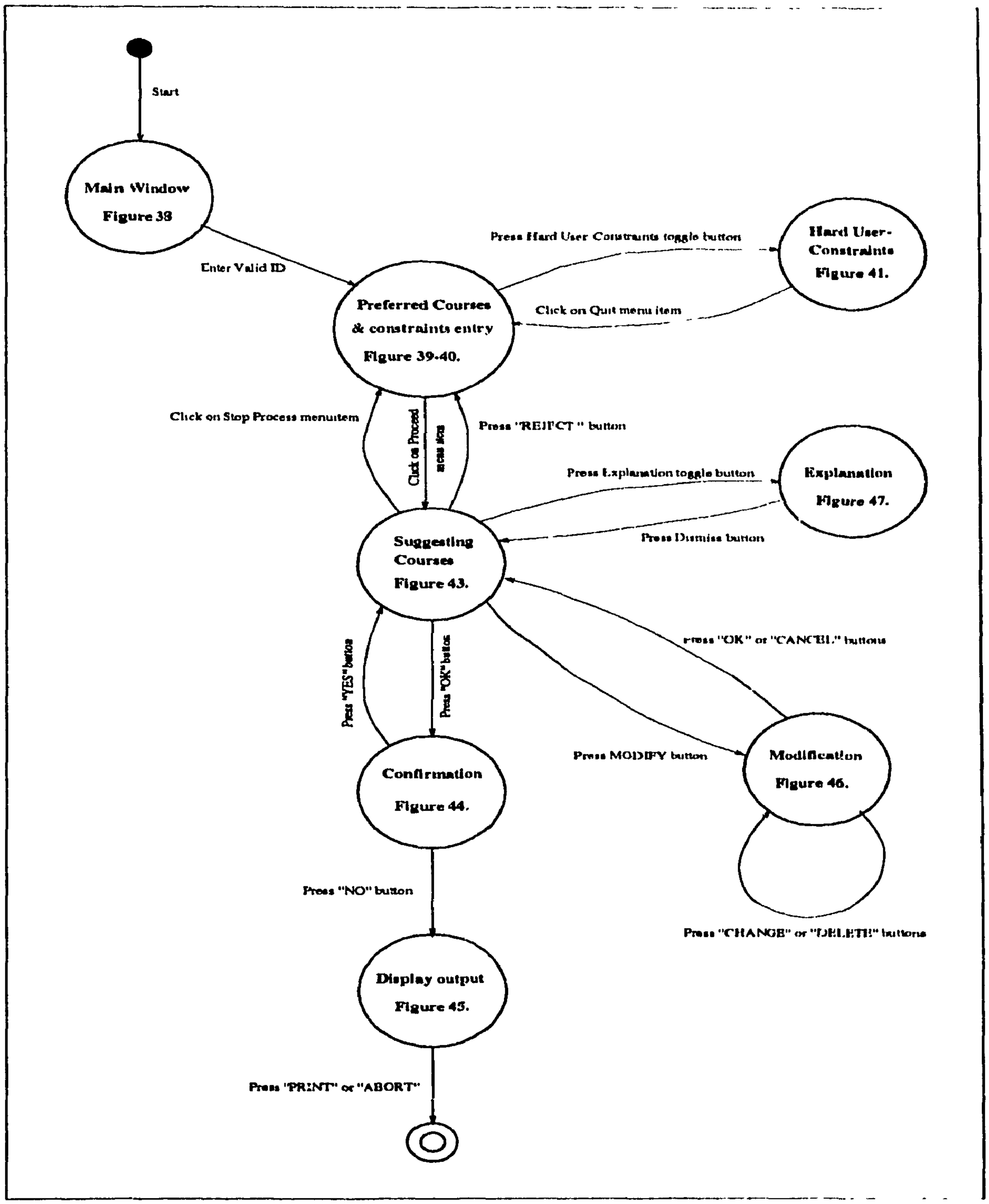


Figure 49: State Diagram of the Graphical User Interface

4.2.2 Communication layer between CAS and its GUI

This project was carried out by two students as two parts: the first part contains the design and implementation of CAS [Duo], and the second part the design and implementation of the Graphical User Interface is presented in this major report. CAS and GUI communicate through the Communication layer as shown in Figure 50.

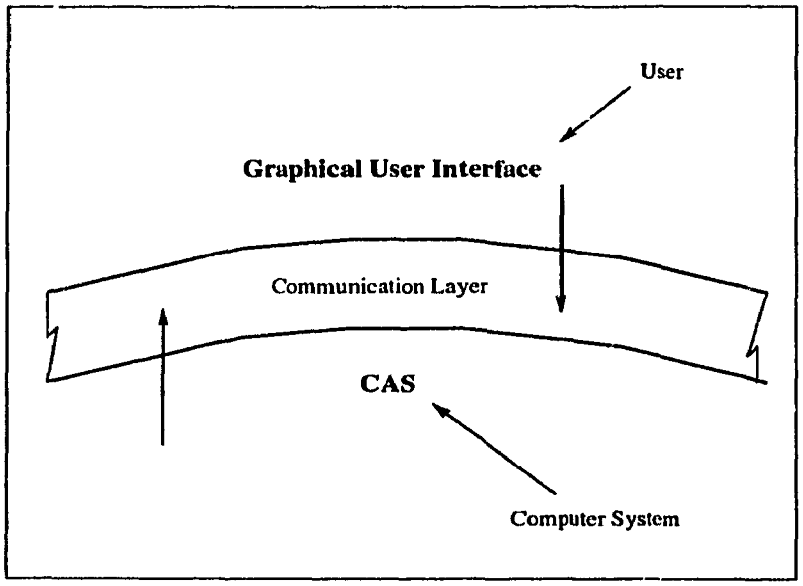


Figure 50: Communication between CAS and its interface

Communication layer is positioned between CAS and its user interface. It contains a set of function definition, and declaration of global object classes. Two **User-Constraint** objects (Soft Constraints and Hard Constraints), a **SuggestedCourses** object and an **Explanation** object (See 2.2.2) are made available for the interface as global objects. The interface reads the information from SuggestedCourses (supplies data for "Suggesting Courses") and Explanation (supplies data for "Explanation on Preferred Courses") objects and sends information to CAS by writing into the two UserConstraints objects. Below we summarize the functions along with a brief description as they are seen by the user interface:

ChangeSuggestCourse: Requests the replacement of a course with an other section of the same course (schedule is different). Returns "OK" if the course is changed and "NOT OK" otherwise.

CleanUpCAS: User interface calls it when the user exits the application. (CAS uses this function.)

DelSuggestCourse: Requests the deletion of the specified course from the list of suggested courses (SuggestedCourses object). Returns "OK" if the course is deleted from the list and "NOT OK" otherwise.

InitializeCAS: It is called from the Control function at start up of the application. CAS uses it to initialize global objects.

RequestCourseSched: Requests the schedules of all available sections of a certain course. Returns "OK" if the course is valid and the schedule list is available, "NOT OK" otherwise.

ResetAdvise: Informs CAS that the advising session has been terminated.

StartAdvise: This function instructs CAS to select courses for the student. If CAS has produced feasible list of courses, value returned is "OK". The list of suggested courses and the explanation are stored in SuggestedCourses and Explanation global objects respectively.

StopAdvise: This function instructs CAS to stop creating the list of suggested courses and explanation, clear Soft Constraints object.

SuggestMoreCourse: Requests additional courses to be added (suggested) to the list of suggested courses. Returns "OK" if the requested suggestion has been finished successfully, and "NOT OK" otherwise.

ValidateID: UI sends student ID to CAS. This function returns "OK" if ID is valid and "NOT OK" otherwise.

The design of CAS user interface is based on the Dynamic Model of CAS described in Section 2.2.3. The Global Event Flow diagram was used to determine the user interface component classes. The background computation is based on the State Diagrams of CAS described in section 2.2.3. It can be seen that for each scenario given in Dynamic Model there is a corresponding path in the GUI that one can walk through.

Chapter 5

Summary

This major report is a contribution to the development of GUI using a class library based on C++ and OSF/Motif. Reusable Module Library (RML) is a class library, developed as a part of this work and its use is demonstrated through a case study. The case study is a project called CAS (Course Advisor System). CAS is divided into two parts: Functional part is considered by another student, GUI part is the focus of this project. Throughout the design of CAS as well as the class library RML, Object Oriented techniques are followed.

RML classes are objects of higher complexity, consisting of Motif widgets and gluing C++ code. From the analysis of CAS, we have generalized a class of User Interfaces that will have the following characteristics: " Interactive input and Iterative Specification of the user needs. (i.e. partial need is specified - observe the output - modify the specification.)"
Such GUI s have the following needs:

- Enter a list of items
- Examine a list of items
- Select one or more items from a list
- Interactively relax or tighten constraints (after browsing through a partial solution)
- Menu based selection (pull-down)
- Pop-up dialog boxes

The RML classes developed in this work satisfy the above needs. The user interface of CAS is implemented using C++ and RML. Figure 48. shows the Component Graph of the user interface. We can conclude that the collection of RML classes provides sufficient number of features to build practical graphical interfaces. Many of the RML classes provide hooks for deriving new additional modules or extending a module. For example, **List** object class contains two empty **RowColumn** widgets (see Figure 23. (b)). RowColumn is a manager widget that can hold an arbitrary arrangement of widgets and other RML classes. Using this as a hook the RML class **List** can be extended. The main purpose of RML is to prototype a GUI without having any additional learning overhead for programmers.

Appendix A

Scenarios

1.

The CAS asks the Student to type in his/her student ID#.
The Student enters his/her student ID#.
The CAS reads Student's transcript.
The CAS determines Student's Status, Option, Date of First Registration, from the transcript.
The CAS obtains Degree Requirement (based on his/her Option and Date of First Registration)
The CAS extracts the courses already taken from the transcript.
The Student enters the courses he/she prefers to take.
The Student enters his/her preferences and constraints regarding: campus, workload , time, courses that he/she does not want to take.
The Student instructs CAS to perform advising according to the provided information.
The CAS reads the courses the student prefers to take and other preferences or constraints (determines if there has been entered any).
The CAS examines the courses Student prefers to take if they are qualifying courses (Student can take them).
The CAS selects the qualifying courses from Student's preference list w.r.t. his/her preferences and constraints.
The CAS selects additional qualifying courses to satisfy required workload w.r.t. his/her preferences and constraints. The CAS prepares explanation on Preferred courses.
The CAS displays suggested courses.
The Student requests explanation on Preferred courses.
The CAS displays explanation. The **Student confirms the set of suggested courses.**
The CAS asks if Student needs more courses.
The Student wants more courses.
The Student may enter additional preferences and constraints regarding: campus, workload , time, courses that he/she does not want to take.
The Student instructs CAS to perform advising according to the provided information.
The CAS reads and evaluates the Student's preferences and constraints. The CAS selects additional courses.
The CAS displays suggested courses.
Student confirms the set of suggested courses.
The CAS asks if Student needs more courses. The Student does not want more courses. The CAS prints Suggested Courses with their schedule and location.
The CAS asks the Student (next student) to type in his/her student ID#.

2.

The CAS asks the Student to type in his/her student ID#.
The Student enters his/her student ID#.
The CAS reads Student's transcript.
The CAS determines Student's Status, Option, Date of First Registration, from the transcript.
The CAS obtains Degree Requirement (based on his/her Option and Date of First Registration).
The CAS extracts the courses already taken from the transcript.
The Student instructs CAS to perform advising according to the provided information.
The CAS reads the courses the student prefers to take and other preferences or constraints (determines if there has been entered any).
The CAS selects qualifying courses to satisfy required workload w.r.t. Student's preferences and constraints (if there is any).
The CAS displays suggested courses.
The Student requests explanation on Preferred courses.
The CAS displays message to Student, "No preferred course has been entered. Explanation is not prepared."
The Student confirms the set of suggested courses.
The CAS asks if Student needs more courses.
The Student does not want more courses.
The CAS prints Suggested Courses with their schedule and location.
The CAS asks the Student to type in his/her student ID#.

3.

The CAS asks the Student to type in his/her student ID#.
The Student enters his/her student ID#.
The CAS reads Student's transcript.
The CAS determines Student's Status, Option, Date of First Registration, from the transcript.
The CAS obtains Degree Requirement (based on his/her Option and Date of First Registration).
The CAS extracts the courses already taken from the transcript.
The Student instructs CAS to perform advising according to the provided information.
The CAS reads the courses the student prefers to take and other preferences or constraints (determines if there has been entered any).
The CAS selects qualifying courses to satisfy required workload w.r.t. Student's preferences and constraints (if there is any).
The CAS displays suggested courses.
The Student requests explanation on Preferred courses.
The CAS displays message to Student, "No preferred course has been entered. Explanation is not prepared."
The Student rejects the set of suggested courses.
The CAS records the entire set of suggested courses as constraints (courses Student does not want to take. (Prefix, Course #, Section)
The Student instructs CAS to perform advising according to the provided information.
The CAS reads the courses the student prefers to take and other preferences or constraints (determines if there has been entered any).
The CAS selects qualifying courses to satisfy required workload w.r.t. Student's preferences and constraints (if there is any).
The CAS displays suggested courses.
The Student requests explanation on Preferred courses.
The CAS displays message to Student, "No preferred course has been entered. Explanation is not prepared."
The Student confirms the set of suggested courses.
The CAS asks if Student needs more courses.
The Student does not want more courses.
The CAS prints Suggested Courses with their schedule and location.
The CAS asks the Student to type in his/her student ID#.

4.

The CAS asks the Student to type in his/her student ID#. The Student enters his/her student ID#.
The CAS reads Student's transcript.
The CAS determines Student's Status, Option, Date of First Registration, from the transcript.
The CAS obtains Degree Requirement (based on his/her Option and Date of First Registration).
The CAS extracts the courses already taken from the transcript.
The Student instructs CAS to perform advising according to the provided information.
The CAS reads the courses the student prefers to take and other preferences or constraints (determines if there has been entered any).
The CAS selects qualifying courses to satisfy required workload w.r.t. Student's preferences and constraints (if there is any).
The CAS displays suggested courses.
The Student requests explanation on Preferred courses.
The CAS displays message to Student, "No preferred course has been entered. Explanation is not prepared."
The Student asks for modification.
The CAS displays facility for modification.
The Student may reject one course of the set of suggested courses or may "edit" the set of suggested courses by replacing a course(s) (course 'A') with the same course(s) offered in an other time (course 'B').
The Student signals that modification is ready.
The CAS agrees to perform the modification.
The CAS records the modification as constraint:

Rejected course: indicates that the course is "Unpreferred course" (Prefix, Course #, Section)

Changed course: indicates that course 'A' is "Unpreferred course" and course 'B' is "Preferred course"

The CAS displays suggested courses.
The Student accepts the already suggested courses but instructs CAS to advise additional courses according to the provided information.
The CAS reads the courses the student prefers to take and other preferences or constraints (determines if there has been entered any).
The CAS selects additional qualifying courses (to complement already accepted suggested courses) to satisfy required workload w.r.t. Student's preferences and constraints (if there is any).
The CAS prepares explanation on Preferred courses.
The CAS displays suggested courses.
The Student requests explanation on Preferred courses.
The CAS displays explanation.
The Student confirms the set of suggested courses.
The CAS asks if Student needs more courses.
The Student does not want more courses.
The CAS prints Suggested Courses with their schedule and location.
The CAS asks the Student to type in his/her student ID#.

5.

The CAS asks the Student to type in his/her student ID#.
The Student enters his/her student ID#.
The CAS reads Student's transcript.
The CAS determines Student's Status, Option, Date of First Registration, from the transcript.
The CAS obtains Degree Requirement (based on his/her Option and Date of First Registration).
The CAS extracts the courses already taken from the transcript.
The Student instructs CAS to perform advising according to the provided information.
The CAS reads the courses the student prefers to take and other preferences or constraints (determines if there has been entered any).
The CAS selects qualifying courses to satisfy required workload w.r.t. Student's preferences and constraints (if there is any).
The CAS displays suggested courses.
The Student requests explanation on Preferred courses.
The CAS displays message to Student, "No preferred course has been entered. Explanation is not prepared."
The Student asks for modification.
The CAS displays facility for modification.
The Student may reject one course of the set of suggested courses or may "edit" the set of suggested courses by replacing a course(s) (course 'A') with the same course(s) offered in an other time (course 'B').
The Student signals that modification is ready.
The CAS denies the modification.
The CAS prepares explanation on Preferred courses.
The CAS displays suggested courses.
The Student requests explanation on Preferred courses.
The CAS displays explanation.
The Student accepts the already suggested courses but instructs CAS to advise additional courses according to the provided information.
The CAS reads the courses the student prefers to take and the other preferences.
The CAS examines the courses Student prefers to take if they are qualifying courses (Student can take them).
The CAS selects additional qualifying courses (to complement already accepted suggested courses) to match required workload w.r.t. Student's preferences and constraints (if there is any).
The CAS prepares explanation on Preferred courses.
The CAS displays suggested courses.
The Student requests explanation on Preferred courses.
The CAS displays explanation.
The Student confirms the set of suggested courses.
The CAS asks if Student needs more courses.
The Student does not want more courses.
The CAS prints Suggested Courses with their schedule and location.
The CAS asks the Student to type in his/her student ID#.

6.

The CAS asks the Student to type in his/her student ID#.
The Student enters his/her student ID#.
The CAS reads Student's transcript.
The CAS determines Student's Status, Option, Date of First Registration, from the transcript.
The CAS obtains Degree Requirement (based on his/her Option and Date of First Registration).
The CAS extracts the courses already taken from the transcript.
The Student instructs CAS to perform advising according to the provided information The CAS reads the courses the student prefers to take and other preferences or constraints (determines if there has been entered any).
The CAS selects qualifying courses to satisfy required workload w.r.t. Student's preferences and constraints (if there is any).
The CAS displays suggested courses.
The Student requests explanation on Preferred courses.
The CAS displays message to Student, "No preferred course has been entered. Explanation is not prepared."
The Student asks for modification.
The CAS displays facility for modification.
The Student may reject one course of the set of suggested courses or may "edit" the set of suggested courses by replacing a course(s) (course 'A') with the same course(s) offered in an other time (course 'B').
The Student signals that modification is ready.
The CAS agrees to perform the modification.
The CAS records the modification as constraint:

Rejected course: indicates that the course is "Unpreferred course" (Prefix, Course #, Section)

Changed course: indicates that course 'A' is "Unpreferred course" and course 'B' is "Preferred course"

The CAS prepares explanation on Preferred courses.
The CAS displays suggested courses.
The Student requests explanation on Preferred courses.
The CAS displays explanation.
The Student confirms the set of suggested courses.
The CAS asks if Student needs more courses.
The Student does not want more courses.
The CAS prints Suggested Courses with their schedule and location.
The CAS asks the Student to type in his/her student ID#.

7.

The CAS asks the Student to type in his/her student ID#.
The Student enters his/her student ID#.
The CAS reads Student's transcript.
The CAS determines Student's Status, Option, Date of First Registration, from the transcript.
The CAS obtains Degree Requirement (based on his/her Option and Date of First Registration).
The CAS extracts the courses already taken from the transcript.
The Student instructs CAS to perform advising according to the provided information.
The CAS reads the courses the student prefers to take and other preferences or constraints (determines if there has been entered any).
The CAS selects qualifying courses to satisfy required workload w.r.t. Student's preferences and constraints (if there is any).
The CAS displays suggested courses.
The Student requests explanation on Preferred courses.
The CAS displays message to Student, "No preferred course has been entered. Explanation is not prepared."
The Student asks for modification.
The CAS displays facility for modification.
The Student may reject one course of the set of suggested courses or may "edit" the set of suggested courses by replacing a course(s) (course 'A') with the same course(s) offered in an other time (course 'B').
The Student signals that modification is ready.
The CAS denies to perform the modification.
The CAS prepares explanation on Preferred courses.
The CAS displays suggested courses
The Student requests explanation on Preferred courses.
The CAS displays explanation.
The Student confirms the set of suggested courses.
The CAS asks if Student needs more courses.
The Student does not want more courses.
The CAS prints Suggested Courses with their schedule and location.
The CAS asks the Student to type in his/her student ID#.

8.

The CAS asks the Student to type in his/her student ID#
The Student enters his/her student ID#.
The CAS reads Student's transcript.
The CAS determines Student's Status, Option, Date of First Registration, from the transcript.
The CAS obtains Degree Requirement (based on his/her Option and Date of First Registration).
The CAS extracts the courses already taken from the transcript.
The Student enters the courses he/she prefers to take.
The Student enters his/her preferences and constraints regarding: campus, workload , time, courses that he/she does not want to take.
The Student instructs CAS to perform advising according to the provided information.
The CAS reads the courses the student prefers to take and other preferences or constraints (determines if there has been entered any).
The CAS examines the courses Student prefers to take if they are qualifying courses (Student can take them).
The CAS selects the qualifying courses from Student's preference list w.r.t. his/her preferences and constraints.
The CAS selects additional qualifying courses to satisfy required workload w.r.t. his/her preferences and constraints.
The CAS prepares explanation on Preferred courses.
The CAS displays suggested courses.
The Student requests explanation on Preferred courses.
The CAS displays explanation.
The Student rejects the set of suggested courses.
The CAS records the entire set of suggested courses as constraints (courses Student does not want to take. (Prefix, Course #, Section).
The Student instructs CAS to perform advising according to the provided information.
The CAS reads the courses the student prefers to take and other preferences or constraints (determines if there has been entered any).
The CAS examines the courses Student prefers to take if they are qualifying courses (Student can take them).
The CAS selects the qualifying courses from Student's preference list w.r.t. his/her preferences and constraints.
The CAS selects additional qualifying courses to satisfy required workload w.r.t. his/her preferences and constraints.
The CAS prepares explanation on Preferred courses.
The CAS displays suggested courses.
The Student requests explanation on Preferred courses.
The CAS displays explanation.
The Student confirms the set of suggested courses.
The CAS asks if Student needs more courses.
The Student does not want more courses.
The CAS prints Suggested Courses with their schedule and location.
The CAS asks the Student to type in his/her student ID#.

9.

The CAS asks the Student to type in his/her student ID#. The Student enters his/her student ID#.
The Student aborts the advising session.
The CAS asks the Student to type in his/her student ID#.

10.

The CAS asks the Student to type in his/her student ID#.
The Student enters his/her student ID#.
The Student cancels the present iteration of advising (cancels all courses suggested).
The CAS asks the Student to type in his/her student ID#.

References

[Bra92] Marshall Brain. *Motif Programming: The Essentials...and More.* Digital Press, 1992.

[Duo] Kim Duong. Design and implementation of an intelligent course advisor system. Major technical report, Concordia University, Montreal, in preparation.

[EHJV95] E.Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns, Elements of Reusable Object-Oriented Software.* Addison-Wesley publishing co., first edition, 1995.

[Fou93] Open Software Foundation. *OSF/Motif Style Guide.* Prentice-Hall, 1993.

[Han71] Wilfred J. Hansen. User engineering principles for interactive system. In *Proceedings of the Fall Joint Computer Conference, 39*, AFIPS Press, Montvale, NJ, 1971.

[HF94] Dan Heller and Paula M. Ferguson. *Motif Programming Manual.* O'Reilly & Associates Inc., second edition, 1994.

[Ian92] Renato Iannella. Designing 'safe' user interfaces. *The Australian Computer Journal*, 24(3), August 1992.

[Ltd93] Visual Edge Software Ltd. *Getting Started with UIM/X.* Release 2.5, Document Version 5.93., 1993.

[RBPL93] J. Rumbaugh, M. Blaha, W. Premerlani, and E. Lorensen. *Object-Oriented Modeling and Design.* Prentice-Hall, 1993.

[SG92] Robert W. Scheifler and James Gettys. *X Window System: The Complete Reference to XLIB, X Protocol-X Version 11, Release 5.* Third edition, 1992.

[Shn87] Ben Shneiderman. *Designing the user Interface: Strategies for Effective human-Computer Interfaces.* Addison-Wesley publishing co., 1987.

[Str94] Bjarne Stroustrup. *The C++ Programming Language.* Addison-Wesley publishing co., second edition, 1994.

[Uni94] Concordia University. *Undergraduate Calendar.* Concordia University, 1994.

[You92] Douglas A. Young. *Object-Oriented Programming with C++ and OSF/Motif.* Prentice Hall, first edition, 1992.