CSCD350, Holiday Mailer: Software Requirements Specification

Steve Berg, Zach Lesperance, Tim Lynch, Steven Mather

December 10, 2014

1 Introduction

1.1 Goals and objectives

To build a program that stores addresses of friends and relatives and allows the user to send a 'Holiday Email' to people of her/his choice.

1.2 Software context

Consumer grade product that is intuitive for use by the average computer user. The software handles their 'Holiday Letters' for them electronically.

1.3 Major constraints

The software will need to be able to run minimally on a consumer grade machine so as to not noticeably impact performance. Web connection while running to allow access to the external e-mail servers.

1.4 Use-cases

Use case Add new person to database

Actor user
Basic choose to enter new person, enter first name, enter last name, enter address, enter if and email was sent, submit to database

Use case Remove person from database

Actor user
Basic choose to remove, choose who to remove, make sure, remove

Use case Display entries by last name

Actor user
Basic choose to display, choose to display by last name, display

Use case Display entries by first name

Actor user
Basic choose to display, choose to display by first name, display

Use case Display entries by last received date

Actor user
Basic choose to display, choose to display by last received date, display

Use case Display entries with last name with certain letter

Actor user
Basic choose to display, choose to display by certain letters, display

Use case send email to everyone

Actor user
Basic choose to send email to everyone, pick holiday email to send, send

Use case send email to everyone who has sent one

Actor user
Basic choose to send email to everyone who has sent you one, pick holiday email to send, send

Use case send email to specific people

Actor user
Basic choose to send email to specific people, pick people, pick holiday email to send, send

Use Case Diagram

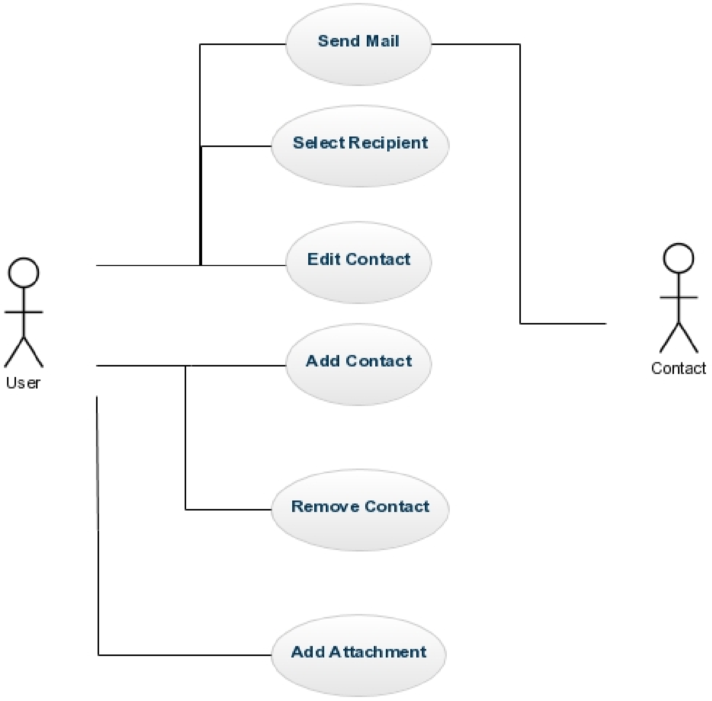

Figure 1: Use Case Diagram

2 Data Model and Description

For this project, we will be using SQLite technology for our data object interactions.

2.1 Data Description

2.1.1 Data objects

Name: contacts
Description: A table that stores all of the user's contacts and their information
Fields:

- email (varchar(50)): The contact's email address
- firstName (varchar(50)): The contact's first name
- lastName (varchar(50)): The contact's surname
- lastReceivedYear (int): The year that the contact last sent a holiday letter to the user

2.1.2 Complete data model

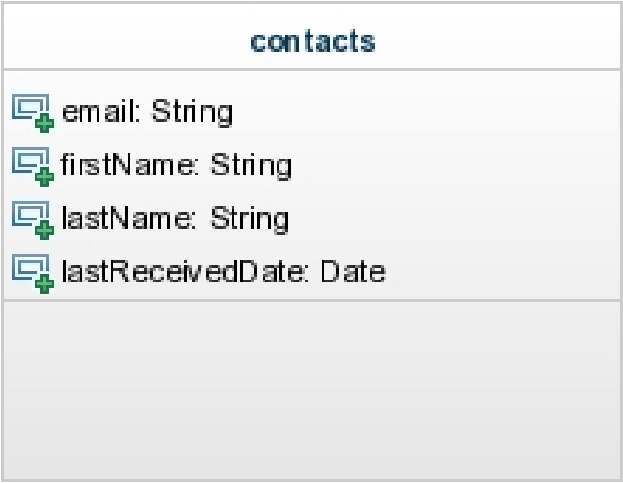

Figure 2: Entity-Relation Diagram for our Data Model

3 Functional Model and Description

The Client class is the main anchor for the program.
It contains references to two objects that handle all input from the user and all output to the user.
The client will also contain the methods needed to delegate work to the proper objects, i.e. contact creation, contact modification, contact deletion, display.

The DBAccess object will be used for all interaction with the database.
The MailControl object track any additions and removals and save them to the database using DBAccess.
The contact class is an object that represents tuples in the database.
The userOut object handles the interactions the program has with the user.

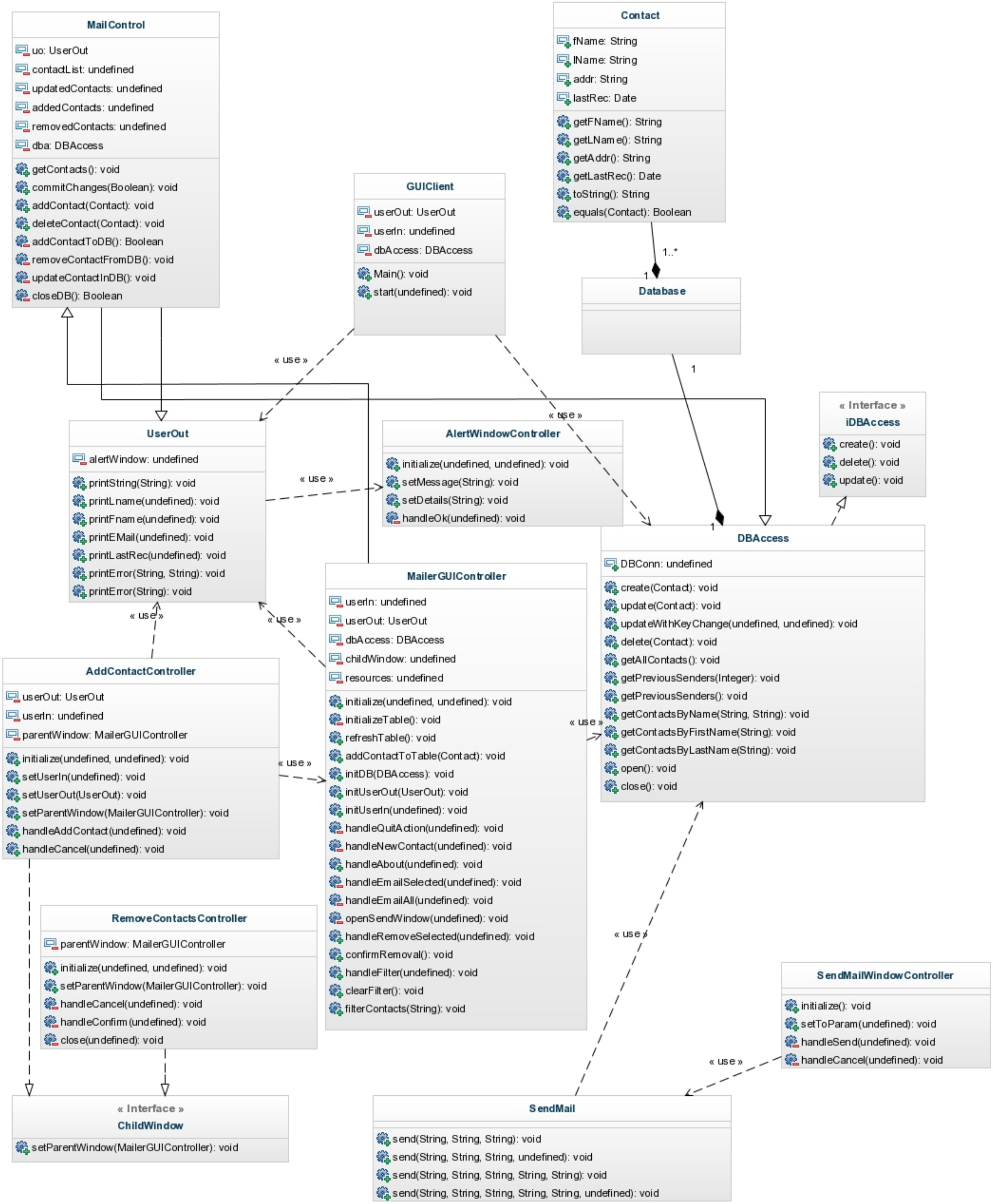
MailControl
uo: UserOut
contactList: undefined
updatedContacts: undefined
addedContacts: undefined
removedContacts: undefined
dba: DBAccess
getContacts(): void
commitChanges(Boolean): void
addContact(Contact): void
deleteContact(Contact): void
addContactToDB(): Boolean
removeContactFromDB(): void
updateContactInDB(): void
closeDB(): Boolean
Contact
fName: String
lName: String
addr: String
lastRec: Date
getFName(): String
getLName(): String
getAddr(): String
getLastRec(): Date
toString(): String
equals(Contact): Boolean
GUIClient
userOut: UserOut
userIn: undefined
dbAccess: DBAccess
Main(): void
start(undefined): void
1..*
1
Database
1
« use »
« use »
« Interface »
iDBAccess
create(): void
delete(): void
update(): void
UserOut
alertWindow: undefined
printString(String): void
printLname(undefined): void
printFname(undefined): void
printEMail(undefined): void
printLastRec(undefined): void
printError(String, String): void
printError(String): void
« use »
AlertWindowController
initialize(undefined, undefined): void
setMessage(String): void
setDetails(String): void
handleOk(undefined): void
1
DBAccess
DBConn: undefined
create(Contact): void
update(Contact): void
updateWithKeyChange(undefined, undefined): void
delete(Contact): void
getAllContacts(): void
getPreviousSenders(Integer): void
getPreviousSenders(): void
getContactsByName(String, String): void
getContactsByFirstName(String): void
getContactsByLastName(String): void
open(): void
close(): void
« use »
« use »
MailerGUIController
userIn: undefined
userOut: UserOut
dbAccess: DBAccess
childWindow: undefined
resources: undefined
initialize(undefined, undefined): void
initializeTable(): void
refreshTable(): void
addContactToTable(Contact): void
initDB(DBAccess): void
initUserOut(UserOut): void
initUserIn(undefined): void
handleQuitAction(undefined): void
handleNewContact(undefined): void
handleAbout(undefined): void
handleEmailSelected(undefined): void
handleEmailAll(undefined): void
openSendWindow(undefined): void
handleRemoveSelected(undefined): void
confirmRemoval(): void
handleFilter(undefined): void
clearFilter(): void
filterContacts(String): void
« use »
AddContactController
userOut: UserOut
userIn: undefined
parentWindow: MailerGUIController
initialize(undefined, undefined): void
setUserIn(undefined): void
setUserOut(UserOut): void
setParentWindow(MailerGUIController): void
handleAddContact(undefined): void
handleCancel(undefined): void
« use »
« use »
RemoveContactsController
parentWindow: MailerGUIController
initialize(undefined, undefined): void
setParentWindow(MailerGUIController): void
handleCancel(undefined): void
handleConfirm(undefined): void
close(undefined): void
SendMailWindowController
initialize(): void
setToParam(undefined): void
handleSend(undefined): void
handleCancel(undefined): void
« use »
« Interface »
ChildWindow
setParentWindow(MailerGUIController): void
SendMail
send(String, String, String): void
send(String, String, String, undefined): void
send(String, String, String, String, String): void
send(String, String, String, String, String, undefined): void

Figure 3: UML Class Diagram of the Software

3.1 Software Interface Description

3.1.1 Human interface

A command line menu driven interface will initially be built and if time permits and we make all the properties that we want, a graphical user interface will be build.

4 Behavioral Model and Description

A description of the behavior of the software is presented.

4.0.2 States

State: IDLE
Description: Program window is open and waiting for user interaction.

State: Add Contact
Description: User has requested to add a contact, the add contact form appears, the user fills it out and submits it, returning to IDLE.

State: Select Contact
Description: User has selected one or more contacts from the table of available contacts.

State: Remove Contact
Description: User has selected to remove contact. User is alerted that this action cannot be undone. User can accept or cancel the request, either returning the program to IDLE.

State: Add Attachment
Description: User has selected one or more contacts to receive an email and has chosen to add an attachment. User can select a file on their local file system to attach.

State: Send Mail
Description: User has chosen to send emails to one or more contacts. The system sends the mail and returns to IDLE.

4.1 State Transition Diagrams

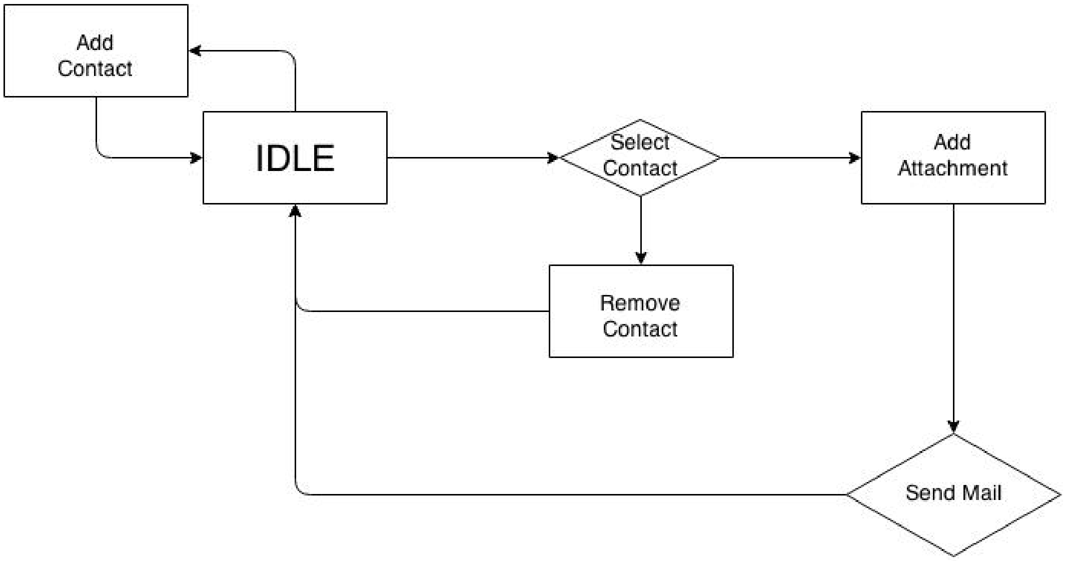

Figure 4: UML State Diagram of the Software

Sequence Diagram

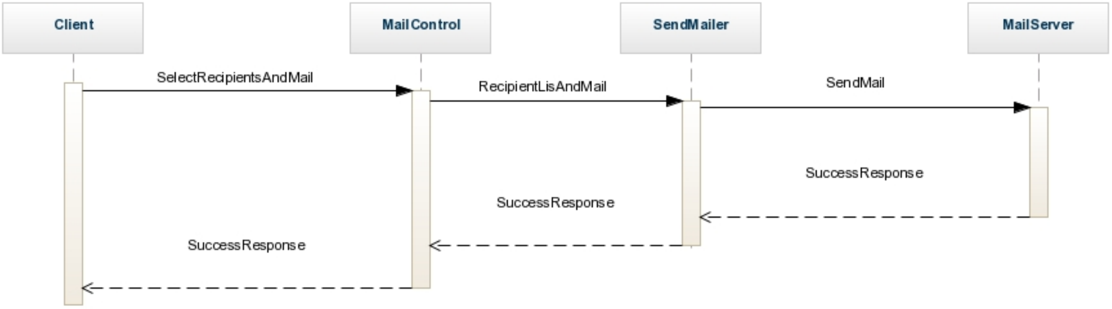

Figure 5: Sequence Diagram of the Software

5 Validation Criteria

5.1 Classes of tests

We will be running JUnit tests against each iteration of our code to make sure that our program stays consistent.

JUnit will be testing:

- All CRUD (Create Update Delete) methods, database interation, data validation, etc methods.
- The ability to send mail
- User input functions