Austin Osborn
COMP 2710-001
Lab 2 Design Portion (Revised)
Aeo0015@auburn.edu

*REVISED COPY:
Summary of revisions/Notes:

- Initial Execution now only gives user option to enter a user name. If they already exist, they are logged in. If not a new user is created and that new user is logged in as active. This matches closer to the Lab 2 doc.
- New user and switch user are still fully functional and function similarly to lab 1. These are not required, but I kept them anyways, as they still work.
- Class diagram has been modified slightly to better suit the system requirements. Some changes include…
    - Removal of type identifier for Message class (wasn't used)
    - Message class now has an method (sortMessage) that uses an algorithm to sort messages which are to be printed by time.
    - User objects now have a list of all groups of which they are a member.
    - Some system methods have been renamed and reparametrized as needed.
        - Focus on breaking down methods into more reusable parts and making code more readable.
    - MessageBuffer now contains combo functions to parse and print wallpages and homepages. Parsing is done in each method as appropriate to collect the appropriate messages.
    - System now has method to create new files.

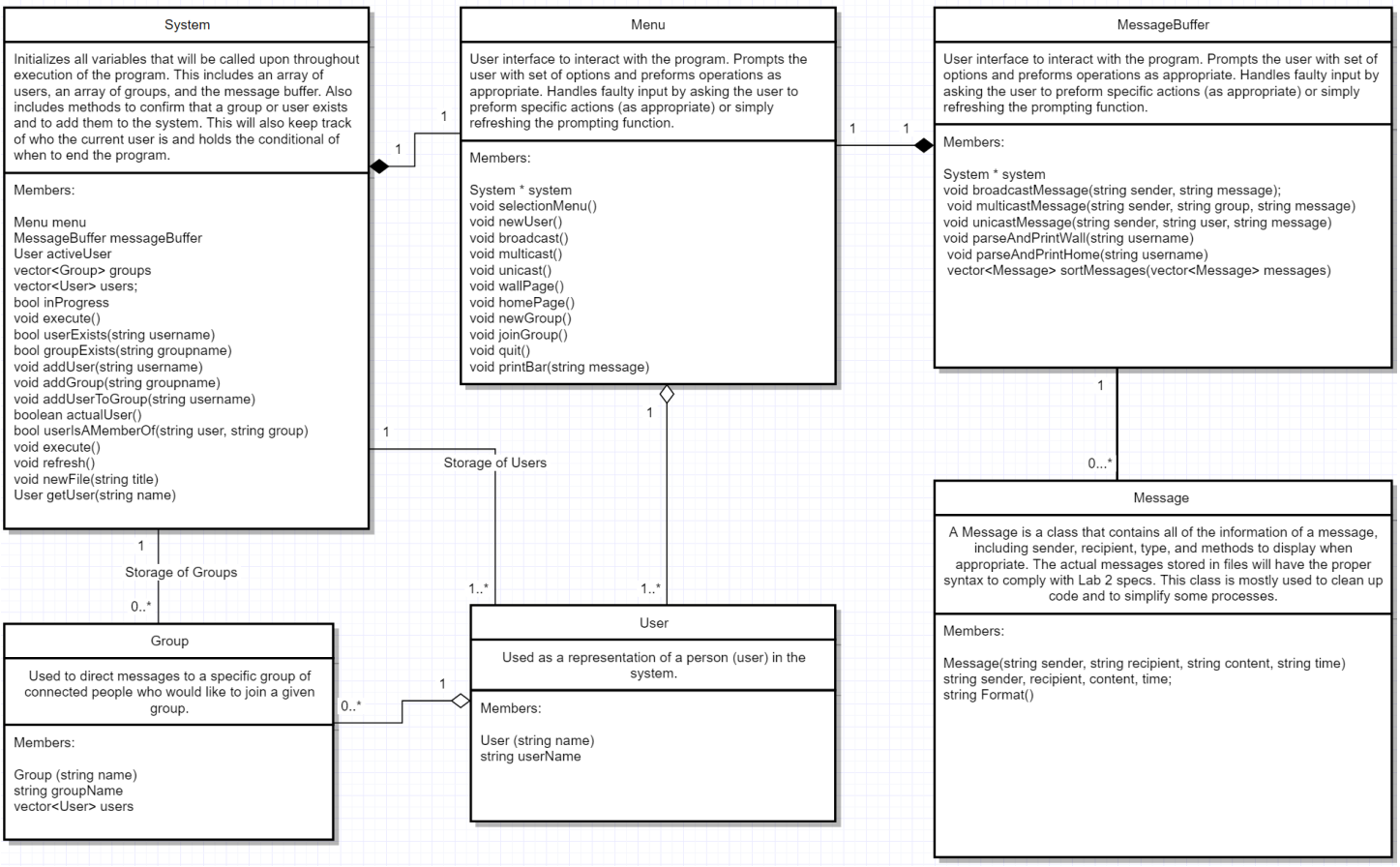
Analysis (Use Cases)

- **Initial execution:** The user is greeted with a greeting message and presented with a menu of options (selection prompt). The first thing they should do is create a new user, as all other options in the menu are contingent upon at least one user being present.
- **Create a new user:** The user selects this option from a list of options. They then enter a name. This user is now logged in as the active user.
- **Send a message:** The user selects one of three options (broadcast, multicast, or unicast) depending on who they want to send the message to. They are then prompted to enter the recipient user or group (as appropriate). If the recipient is valid, then they can enter a message and end their message by going to a new line and typing "^!". Message is now stored and usable in the system going forward.
- **Display a wall page:** The user wants to select this option to see a list of all messages they have sent. The system will show the user up to 2 messages they have sent, and if there are more asks the user if they want to display all.
- **Display a home page:** The user wants to select this option to see a list of all messages directed at them. This includes all broadcast messages, any multicast message for which they are a member of the group, and any unicast message directed at them. The system

will show up to 2 messages, and if there are more will ask the user if they want to display all.

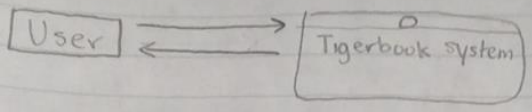
- **Create a group:** The user selects this option to create a new group that users can join. The user is prompted to enter a name for this new group. If it doesn't already exist, it's created, otherwise the user is alerted that their input is invalid and directed back to the selection prompt.
- **Join a group:** The user selects this option to add themselves to an already existing group. They are prompted to enter the name of the group they would like to enter. If this group exists, they are added to the group. Otherwise, they are told the group doesn't exist.
- **Switch users:** The user would like to change from one User to another User. They are prompted to enter the name of the User they would like to switch to. If it exists, the active User is changed. If not, the user is greeted once again with the selection prompt.
- **Quit:** The user would like to close the program. They choose this option and the program will terminate.

Class Diagram

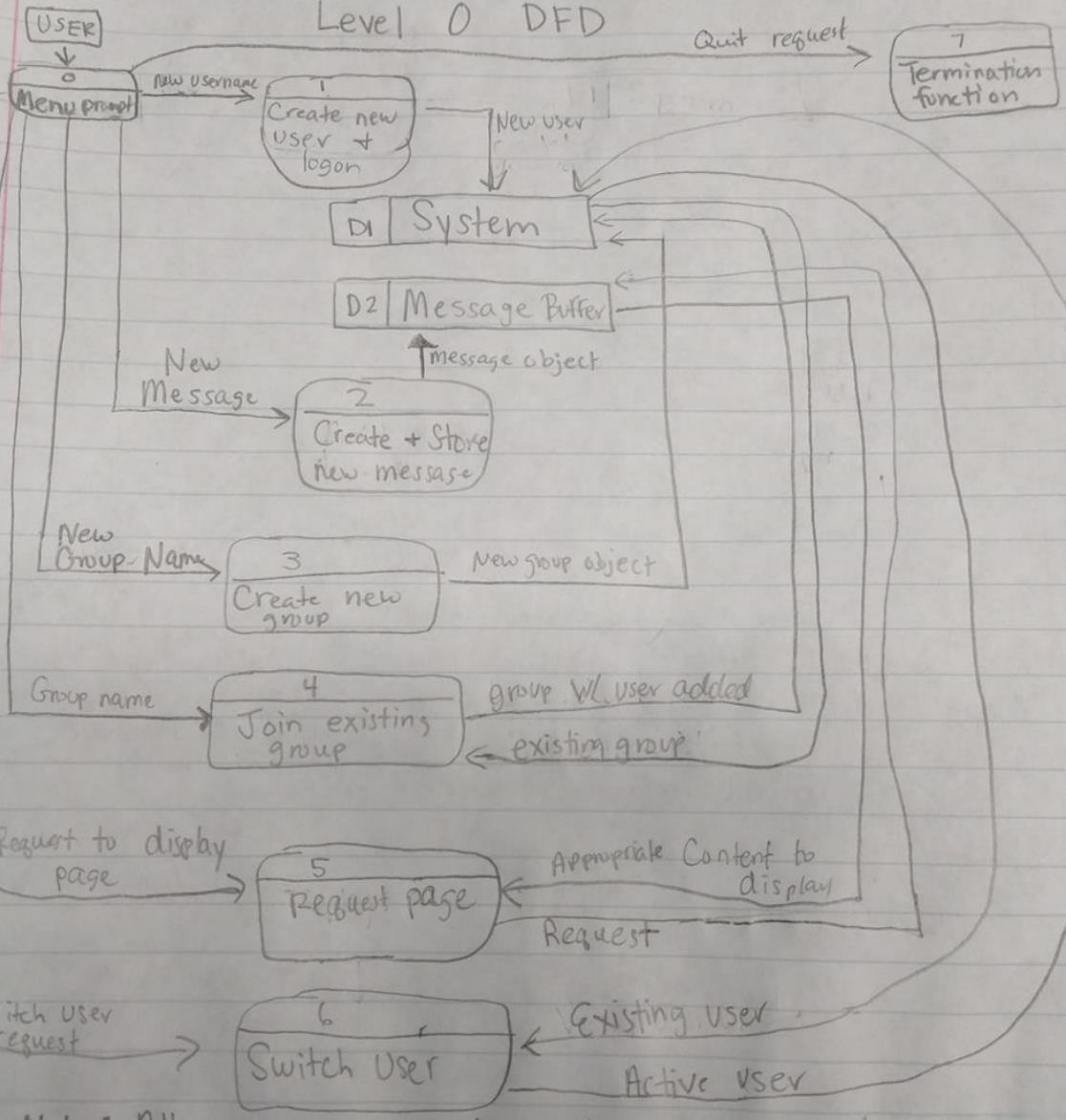


Data Flow Diagrams

Context Diagram

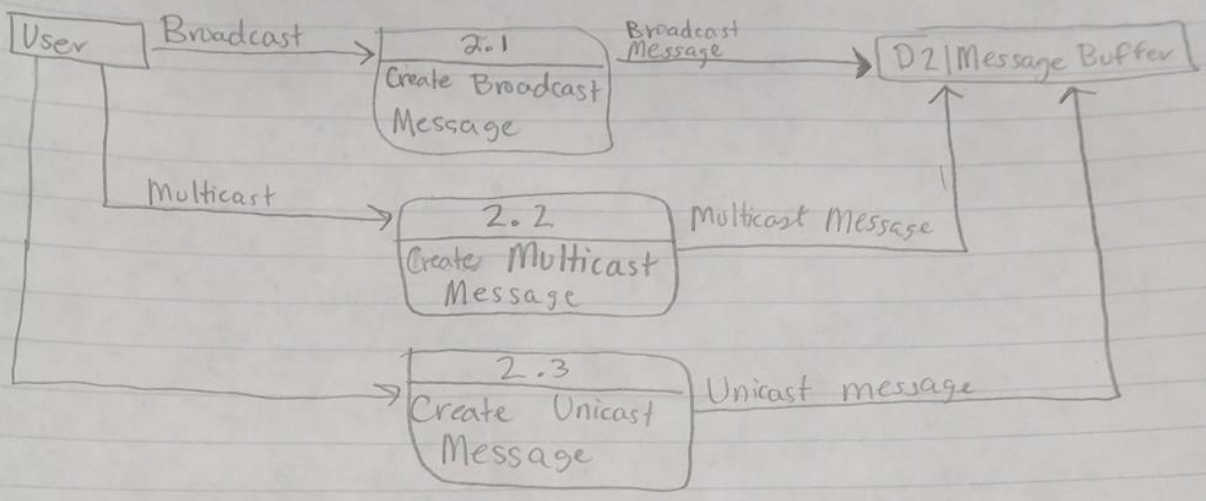


Level 0 DFD

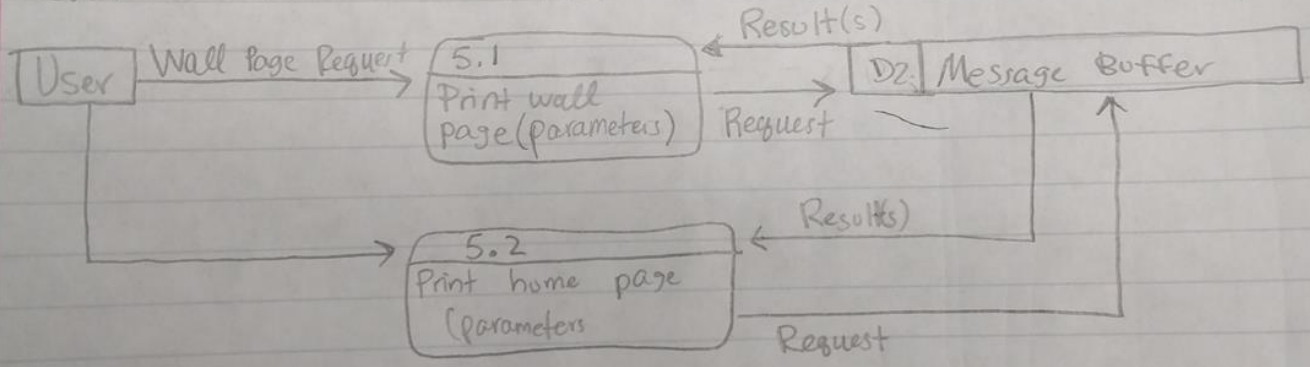


* Note: All processes end with returning to menu prompt, with exception of process 7.

Level 1 DFD for Process 2



Level 1 DFD for Process 5



Testing (Test Cases)

*Test cases are presented here by describing what I will be checking for. I mention what the technical side of the system will be doing as well as the expected console output given any input. During testing, I will verify files create and are formatted properly.

A. System Wide

1. Program is first executed.
   a) MessageBuffer class is initialized. Note, messages are read from and stored in the directory of execution as .txt files. The files in there are to conform with the formatting specified in the Lab 2 handout.
   b) After MessageBuffer object is configured, the system will initialize users and groups from parsing through SYSTEM_ALL_USERS.txt and SYSTEM_ALL_GROUPS.txt. If these files don't exist, they are created. If they do exist, they are read from.
   c) Program should display box containing "Welcome to tigerbook social network"
   d) Menu should prompt a username and sign that user in.
      - If they exist, log them in.
      - If they don't, the system adds a new user and makes them active.
2. User inputs a string on menu prompt and presses enter
   a) If input invalid, system should ask again for valid input
   b) If input is valid, behavior is proper for whatever function is called (specific examples will be listed in part B of "Testing (Test Cases)"
3. Any action is performed.
   a) System refresh() function is called to ensure that the system is synchronized with any and all other concurrent sessions of Tigerbook being executed at once.
      - This testing will involve having multiple instances open at once, going back and forth between them, and checking for continuity in both sessions.

B. Specific Objects/Functions

1. Broadcast message request sent
   a) Prompts user to enter their message
   b) MessageBuffer accesses file "_All.txt".
      - Message_Buffer should append a string in the following format: "{!systemtime!}<!activeusername!>message_here" with no errors. Uses messageBuffer member broadcastMessage(string username, string message) to do this.
2. Mutlicast message request sent
   a) Prompts user to enter receiving group
      - If group doesn't exist, then user is told this and reprompted.

b) Prompts user to enter their message
c) Message_Buffer should add a message in the following format: "{!systemtime!}<!user_name_here!>message_here" with no errors. Uses messageBuffer member to open appropriate group name txt file and multicastMessage(string sender, string group, string message) to do this.

3. Unicast message request sent
   a) Prompts user to enter receiving user
      - If user doesn't exist, the user is told this and the program should restart menu selection prompt.
   b) Prompts user to enter their message
   c) MessageBuffer of the system accesses the appopriate username.txt file and appends a string to the start of the file. This string will be made from unicastMessage(string sender, string user, string message) where the user parameter will be the receiving user's username.
4. Create group request sent
   a) User is prompted to input name of their new group
   b) System refresh is called to pull in most updated set of groups.
   c) System member containsGroup(string groupName) is called.
      - If returns false, then addGroup(string groupname) is called.
        - SYSTEM_ALL_GROUPS.txt is accessed. New line is made and groupname is added followed by a colon and no other characters, as the group will have no members.
      - If returns true, then user is told group already exists and is reprompted to menu selection.
5. Join group request sent
   a) User is prompted to input name of group they wish to join
   b) System refresh is called to pull in most updated set of groups.
   c) System member containsGroup(string groupName) is called.
      - If returns true, then user is added to group with that name.
        - SYSTEM_GROUPS.txt is accessed. Line containing groupname has a " USERNAME|" appended to it. File is saved.
        - SYSETEM_USERS.txt is accessed. Line containing current user has a "GROUPNAME|" appended to it.
      - If returns false, then user is alerted that the group does not exist and taken back to selection prompt.
6. Request wall page sent
   a) printAndParseWall(string username) is called with System.activeUser.username used as parameter.
      - Method should output first two messages of which the user is sending (ie, <!username part)
      - Method will then prompt user asking if they would like to display all.
7. Request home page sent

a) parseAndPrintHome(string username) is called with System.activeUser.username used as parameter
    - Method should output first two messages either unicasted to them, containing a group of which they are a part of (except for messages sent by themselves), and any broadcast messages
    - Method should then prompt user asking if they would like to display all.

8. Switch user request sent
    a) User should be prompted to enter the name of the User they would like to switch to
    b) System method containsUser(string username) is called.
        - If returns false, user is alerted user does not exist.
        - If true, system selects appropriate user to set as active in the system.
9. Quit program request sent
    a) System member inProgress is set to false and the main while loop in the System::execute() is broken.
    b) Program exits gracefully with exit code 0.