UNIVERSAL MESSENGER : TOP LEVEL DESIGN

REQUIREMENTS :

UM is a chat client that allows the user to send messages to people who use ICQ clients, AOL Instant Messenger Clients and MSN messenger clients.

Common Features for communicating with all protocols :

- Simple message transfer to buddies– there will be no support for different fonts, colors, nor support for bold or italicized text.
- Simple message transfer to *non* buddies.
- In both of the above cases, the user must be able to tell if the message was sent successfully or not.
- File transfer – Sending files to users of the different chat protocols programs. The GUI will indicate the progress of the file transfer procedure for the duration of the transfer –i.e. file size, and "% complete so far".
- Change Online status : Online, Offline, Away, Busy – other status indicators will be approximated to these.
- Message history – Any messages sent to or from the user will be stored for future retreival. These messages can be deleted if need be. The GUI must provide an interface for viewing old messages.
- Online buddy status– The user should be able to see what the online status of the buddies on the buddy list are.
- Note - There will be no support for multiple user chats.

ICQ Features:

- Send URL

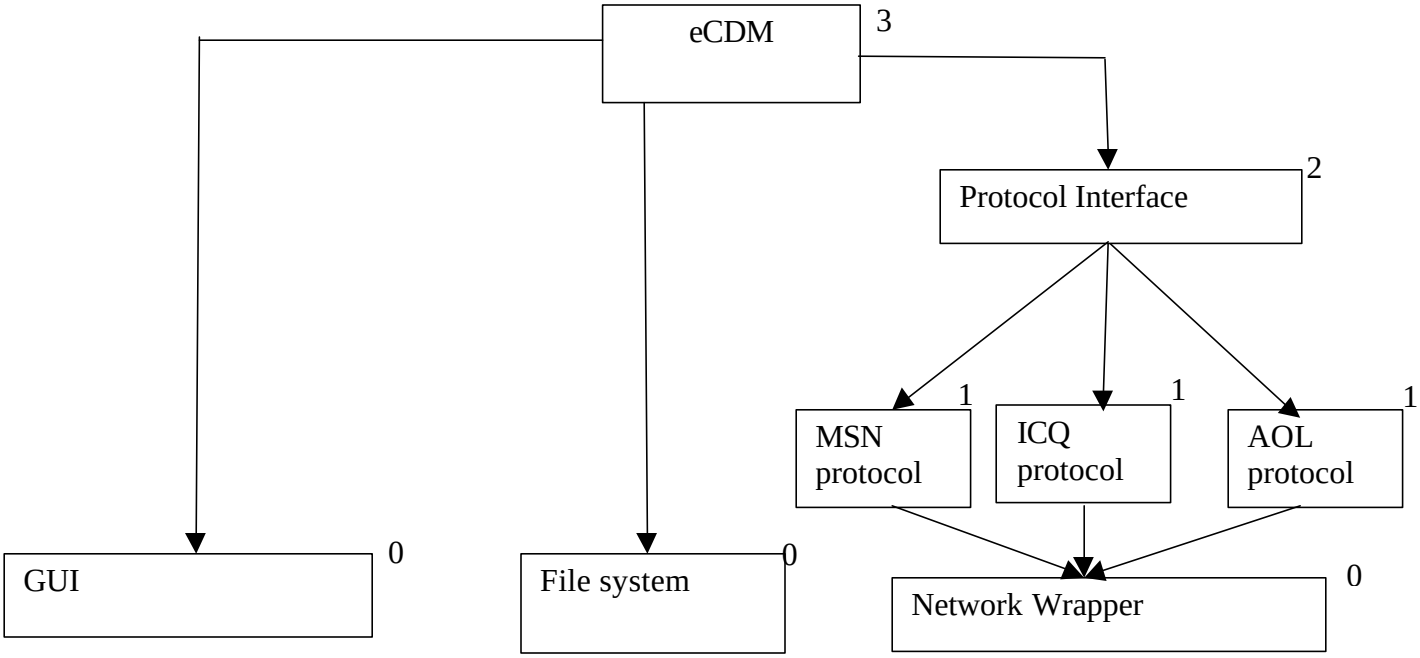
AOL features :

- The user should be able to warn buddies.
- Warnings and rate limitations – when the user is "warned", the server sends a message to the user indicating this. When the warning level reaches some level, the server limits the number of messages that the user is allowed to send within a certain time period. This is also a function of the network traffic at that point. The GUI is required to display a "message not sent" error, and outline that the problem was related to the rate limit.

MSN features :

- User is typing you a message - Basically the most attractive feature of MSN. When you are having a chat session in MSN, the GUI is required to display whether or not, the buddy is in the process of typing a message to the user. Note that the Protocol Interface must allow for this.

LEVELIZED COMPONENT DIAGRAM :



MODULE DESCRIPTIONS :

ECDM : ESCALATED CYCLIC DEPENDENCY MANAGER –

This is the top level of the program.
It will instantiate the GUI and the Protocol interface components. These two components will talk to the eCDM through callbacks. The callbacks will be implemented using the QT signal/Slot mechanism.
The GUI, ECDM and Protocol Interface classes will all run in the same thread.
The individual Protocol classes will run in separate threads.
Important callbacks provided by the ECDM:

GUI: (i.e. the GUI will tell the ECDM this)

- logon
- send message to buddy
- add a buddy
- delete a buddy
- send a file to a buddy

Protocol Interface : (i.e. the Protocol will tell the ECDM these)

- Buddy sends a message to the user
- User is warned by the another user (AOL specific)
- User logon successful/unsuccesful
- User disconnected from server
- Buddy sends file to the user

GUI :

The primary function of the GUI is to provide an interface to the system. However, this system is heavily driven by the GUI, so system functionality is dependent on the GUI. Also the success of the project is dependent on how easily users can relate to the GUI.
Signals to the ECDM :

- ChangeOnlineStatus (between Online/Offline/Not Available/Busy)
- sendMessageToBuddy
- addBuddy
- removeBuddy
- BuddyListRequest
- SendFile

So as not to tie up the GUI while the ECDM is communicating with the network, the GUI will have “slots” associated the above signals. The ECDM uses these slots to notify the GUI that these operations are completed.
(Design Note : This allows the GUI the flexibility to block or not block when waiting for a response. For example, if the GUI emits a “sendMessageToBuddy” signal, the GUI then has the choice of :

- waiting for a confirmation that the message has been received, in the meantime holding up the GUI.

- Allowing the user to send messages to other buddies, and inform the user of success later, thus not blocking the GUI.

The ECDM and the GUI actually reside in the same thread.

FILE SYSTEM INTERFACE

This system provides the ECDM with a user profile manager. If multiple people are using the client and do not wish to reenter all their names and passwords everytime they use the system, they can create a "profile". A Profile stores the login name and password of any protocol the user wants it to and also stores a message history.

Methods that the file system interface must provide:

- createNewProfile : creates a new UM user
- getProfile : Gets the profile of a given user
- GetProfileList : returns a list of all the profiles stored on disk.
- saveMessage : saves messages sent to or from the user to the message history
- addProtocolLogin : saves protocol login information – user name/login number/screen name and password
- getMessageList : allows the retrieval of messages stored on disk that were sent to or from one of the users in the past.

PROTOCOL INTERFACE -

Abstracts out from the ECDM, the protocol specific functionality. The decision of "which protocol to use" given an event from the user is delegated to the Protocol Interface.

The Protocol Interface, resides in the same thread as the ECDM. Note that since this is a layer between the ECDM and a potentially very diverse set of protocols, this interface will have to be provide for a diverse set of methods. For example only the ICQ interface has the concept of "authorizing" users to add you to their buddy list, but we must allow for this in the interface even though only one protocol uses it.

Methods required by the interface:

- Login
- Logout
- ChangeOnlineStatus – online/Not available/busy
- sendMessageToBuddy
- sendFileToBuddy
- getBuddyList – If the protocol queried stores the buddy list online, and not locally, then this function will return that buddy list.
- NeedsAuthorization – returns true, if the protocol has an buddy authorization mechanism.
- RequestAuthorization – Asks the buddy if you can put the buddy on your buddy list – (ICQ specific)

Design discussion :

The main justification for the existence of this layer, is to insulate the ECDM from interfacing with the protocols.

The design team was faced with a choice to either place the protocol classes all in one thread, or allow each protocol class to run in separate threads.
Demoting this functionality by creating the Protocol Interface class, allows us to have a working eCDM that is uncontroversial in design. It lets us, if needed, replace the Protocol Interface class with other implementations, to compare how well each implementation works without affecting the interaction of the ECDM with the GUI and the file system interface.

MSN/ICQ/AOL PROTOCOLS

The heart and soul of the program, the protocol classes will communicate with the appropriate servers (and sometimes buddies directly) to provide the functionality promised in the requirements. The protocols will all by subclasses of an abstract base class, by which the Protocol Interface will refer to them.

The methods provided by the protocol interface(login,logout etc..) would call corresponding functions in the individual protocols.

Also the protocols will to communicate up to the ECDM, signals such as:

- MessageReceivedFromBuddy
- FileReceivedFromBuddy
- Login successful
- Login unsuccessful
- Message send successful
- Message send failed.
- DisconnectedFromServer
- BuddyStatusChanged

NETWORK WRAPPER :

Provides a wrapper around the network so that any operating system specific details are hidden from the Protocol classes and the rest of the program.

GROUP ORGANIZATION

Name (Sun lab login) – role

Aaron (agabow) – Program Manager :
Ensures the project runs on schedule and smoothly and the group members are aware of class related deadlines.

Audrey (syau) – MSN (Microsoft Network) protocol specialist :
Will create a library that is capable of communicating with MSN servers to capture the functionality outlined in the UM network interface.

Jason (jahuang) – System Integrity : System Tester
Will make sure the program is bug free and robust by employing a variety of testing techniques outlined below.

Jon (jdmartin)– ECDM development :
Will implement the ECDM, the "main" component of the program. Details of the ECDM are given below.

Marco(mds) - AIM (America Online Instant Messenger) protocol specialist:
Will create a library that is capable of communicating with AIM servers to capture the functionality outlined in the UM network interface.

Nick (jmoy) - ICQ protocol specialist :
Will create a library that is capable of communicating with ICQ servers to capture the functionality outlined in the UM network interface.

Nigel(ncordeir)– File system interface developer :
Will create the subsystem that creates, manages and stores user profiles, and keeps track of the message history

Shams (mkazi) - Program Architect a.k.a. Archie
Ensures integrity of the design and coding of the system, and ensures that the project's technological objects are met. The coder of last resort.

Shiwon (Shchoe) – User Interface Development Lead
Will implement the interface between the user and the chatting system. The GUI drives the system.

Wolfgang – Network Lead
Will implement the layer between the ECDM and the protocol specific network classes. Also the network layer (layer on top of the network sockets themselves). Will also be responsible for the progress of the development of the protocol classes.

Yazan - Editor
Develops the documentation of the project, including user and technical documentation. The technical documentation aids the future maintainability of the project.

EXTERNAL DEPENDENCIES

- Qt – Apart from our GUI being implemented in QT, we will be using some non-GUI QT functionality in particular callbacks (signals/slots) and file representation (QFile). We justify this by the fact that QT is a portable interface, and can compile under a variety of platforms. It is important to note however, that even if the GUI is replaced QT libraries will be needed to compile the program.
- The ICQ/MSN/AOL IM servers are completely out of our control and information about the future direction of these servers is unavailable. However we are fairly confident that any new improvements to the servers will still leave the server backwards compatible with their older clients hence compatible with our system.
- Licensing issues do not exist given this project is being attempted for academic purposes and not for profit.

TESTING :

1)Component Testing –
Includes rigorous testing of individual component driven by test code that calls all methods in each component individually to check for functionality, limitations and performance.
Steps Include:
-Making sure that all components and methods have been created and can be compiled and called to return dummy values
-Checking for appropriate output from methods. Also checking for upper and lower limits and error messages etc. (e.g. buddy list limitations, message lengths, missing messages (sending a message before logging onto an ICQ server), etc.)
-Testing each class for integrity, memory usage, performance, possible code reviews at this point. (e.g. Does each method do what it should do? Should it do less? More? Can stuff be broken up or combined?)
-Test components for integrity, memory usage, dependencies, adherence to the spec and design docs. (e.g. review design diagrams and stated dependencies and interface, check if components match design and meet requirements)

2)Integration Testing –
- Ensuring that components are communicating with each other correctly, through established interfaces.
Steps Include:
- Testing 2 components together for functionality using test code to fill in any missing components necessary for tests (dummy UI to send input and output to protocol components, and dummy Protocol Interface class, to test the UI)
- Testing more than 2 components together. Separate components should have been rigorously tested at this point.

3)User Testing
- Running the program through the GUI to crash the system. Tests repeat many of the tests before. Steps Include: -Bounds and Limit testing through user input. (e.g. large buddy list again etc.)
-Sanity check on input/output through GUI. (e.g. the GUI doesn't allow bad i/o)
-Running multiple instances of UM on the same machine connecting to a server both using the same protocol to communicate and using different protocols (e.g. ICQ to ICQ messaging or ICQ to AIM messaging)
-Running multiple instances of UM on different machines
-Running UM on a Sun and original ICQ/AIM/MSN clients on a Sun then a Windows box outside of the SunLab
-Running UM on a Sun and multiple ICQ/AIM/MSN clients in addition to other "universal messaging" clients (e.g. jabber, Odigo or whatever it's called)

Strategies
-developing testing tools before components are completed (e.g. random string generation etc)
-listing out the possible limits of the program and how one can break them, then fix them.

-brainstorming all possible test cases and scenerios (e.g. What happens if off by one bug happens here, how much will it affect the rest of the program? Will it completely crash? Or will just a single ICQ message be lost?)
-figuring out where the most important dependencies are, so they can be more carefully tested. -reviewing .h files and interface for changes and potential weak areas
- Memory perturbation – technique that disturbs memory not allocated, to make sure we are not accessing memory we are not supposed to.

(The more creative stuff)
-Offer free food to people in the SunLab to test out UM for 30 minutes at a time and have them report bugs. Possibly setup some stupid contest of sorts to lure beta testers (such as stressed out CS32 students that would rather chat than work on their final project design ;)
-Have the coders of different components read the code of their counterparts and ask questions along the lines of "Why is this done in such and such a way? As opposed to such and such a way?" (e.g. have the GUI coders read, understand and question the protocol code)

Culture :

Our group is awesome, and our project is cool, and that makes development fun. Apart from this however :
The Program Architect will bring food to any meetings.
The group is commissioning the design of a UM logo to build group identity.

SCHEDULE

*****Key date : April 16th Integration******

Week 1&2 :

Friday February 23th – Friday March 9th

Group roster finalized by end of week 1.

Protocol groups : Review description of how each protocol implements : Logon, Message transfer, file transfer. Identify potential problems and obstacles not apparent before. Outline procedures you need others to write.

HANDIN : MARCH 9 : FINALIZED DESIGN DOCUMENT

Week 3 :

Friday March 9th – Wednesday March 14th

GUI Team : Create detailed pictures/drawings of every screen and decide on GUI tools to be used.

Discuss and debate interface issues

Protocol groups: Initiate communication with your servers, start experimenting with logon routines

GUI Team : Create Main Buddy Window

We must have **compilable level 0 interfaces**, and potentially other interfaces.

HANDIN : MARCH 14TH : INTERFACE PROPOSALS , COMPONENT WISE TESTING STRATEGY

Week 4 :

Wednesday March 14th – Monday March 19st Finalize Interface!!

HANDIN : MARCH 19th INTERFACE COMPONENTS

HANDIN : March 23rd : Fully compilable Interfaces, Design Document***

Week 5 : March 19th – March 23rd - :

Protocol and GUI team : **Finish detailed design** if you want to enjoy spring break. We must absolutely start **coding like crazy as soon as we come back.**

SPRING BREAK MARCH 23rd – April 4th

HANDIN : April 4th Detailed Designs

Week 6 and 7 :

April 4th – April 15th

Developers : All the coding must be finished – see that task list for plan of action

Component testing

Week 8:

April 16th : INTEGRATION

Integration testing

Week 9 :

System and User Testing

April 23rd Code review : Print out your code for the world to see!!

*****April 27 DEMO DAY (In class)*****

Week 10 :

May 2nd – DEMO DAY II (World Demo Day)

Week 11 :

HANDIN : May 11th Final Documentation

Project Task list:

This gives a plan of how to approach each task

GUI development :

- **March 12th Rough outline of Header files**
- **March 14th Finalize look and feel by creating diagrams of all screens**
- **Design main screen, with buddies and their online status' shown**
- **Be able to display messages that have been received**
- **Be able to send messages**
- **Display file sending/receiving progress charts**
- **Implement the File system GUI, and allow the user to view message history**
- **Display errors from network**
- **Allow AOL users to warn buddies**
- **MSN user "typing a message" indicator**

Network development :

- **March 12th Outline of Header files**
- **March 15th finalize interfaces**
- **March 16 – Design Network Wrapper and Protocol Interface**
- **Logon process**
- **View buddies online**
- **Receive messages from buddies**
- **Send messages to buddies**
- **Receive files from buddies**
- **Send files to buddies**
- **Send/Receive URL from buddies (ICQ)**
- **Warning level feature implementation (AOL)**
- **User typing a message feature (MSN)**

ECDM development :

- **March 12th Outline of Header files**
- **March 15th Finalize interfaces**
- **Logon process.**
- **Buddy list creation**
- **Changing online status**
- **Transfer of information(messages/files) from network to GUI**
- **Transfer of information(messages/files) form GUI to network**

File system Development :

- **March 12th Outline of Header files**
- **March 15th Finalize interfaces**
- **Creating/deleting User Profiles**
- **Storing user names/passwords**
- **Saving messages to the history**
- **Deleting messages from the history**
- **Retrieval of messages**

Testing :

March 15th march 23th– Review Header files, and design component tests
Unit testing – April 11th-April 16th
Integration testing – April 16st – April 28th
System and User testing.