Csicq — an Applet based ICQ client implementation — project documentation for Internet Applications and Java workshop

Zvika Brakerski[*] Asaf Koren[†]

May 9, 2001

ξ

This document contains documentation for the csicq project. This include project requirements and and design issues up to the detailed design level.

Lower levels of documentation are available separately and of course you can find documentation in the code.

The csicq project also provides a manual for programming using the icqlib package.

[*]zvika@eng.tau.ac.il
[†]asaf@math.tau.ac.il

Contents

1 General information

This paper describes a project that has been implemented as a part of the *Internet Applications and Java* workshop[1] given in the winter semester of the year 2000–2001 in *The school of Computer Science of The Faculty of Exact Science* in *Tel–Aviv University* by Dr. Yosi Arzouan.

The purpose of the project is using advanced new technologies in the field of internet applications to implement a multi-tier application.

2 The project - general description

Our project implements an *Applet based ICQ implementation*, meaning an implementation of an ICQ client over a regular applet that can be run on any browser or virtual machine.

The goal of this project is to enable an average end-user to access the icq network using nothing but a Java enabled web browser. Our user will receive a (basic) GUI and will be able to use all of the basic ICQ features.

3 Problem analysis

3.1 The ICQ network and the ICQ protocol

Serving hundreds of millions of registered users and thousands of new users every day, ICQ is currently the world's largest internet-based personal messaging system.

In essence, ICQ is a client-server application. ICQ servers are owned and maintained by *icq.com Ltd.* which distributes clients to end-users. The end user then uses the client program to contact an ICQ server or other ICQ clients using a certain protocol. This way the user can use the features presented by the ICQ network.

The ICQ protocol is not available to the public. It appears that *icq.com* does not want to reveal it's commercial approach or confine itself to a certain protocol. In spite of that, many projects were aimed at analyzing the ICQ protocol in various methods.[2] The results were used to implement various

[1] Course number 0368.3500.02.

[2] Note that such action is indeed legal, it only requires packet analysis of one's own machine which is in no way prohibited.

unofficial ICQ clients with significant success. Currently there are quite a few open source projects that implement ICQ clients and the "analyzed" version of the protocol is available to the public.

The protocol published is a relatively simple message-sending protocol between the client and the server using both UDP and TCP to maintain connection and transfer information. In addition, a TCP connection between two clients is used to accomplish "private chats".

There is a reason to believe that a client implemented according to the open-source versions of the ICQ protocol will be functional so long as the ICQ network exists. This is because older ICQ clients still follow this protocol and for compatibility reasons it is not likely that it will be changed.

Later in this document we will review the current situation at the ICQ client "market". More information on unofficial ICQ clients can be found there.

3.2 Our approach

Our product is aimed at enabling an indirect connection to the ICQ network using Java Applet technology. Our Applet receives ICQ commands from the user and forwards them to our server which, in turn, pushes it to the ICQ network. The reply from the ICQ network is being returned to the Applet via a TCP connection between the applet and the server.

3.3 Final Product

The final product is composed of two basic applications: a client application and a server application.

The end-user receives a GUI via an Applet (on their web browser or any other Java VM) that sends commands to the server. The server is responsible for providing the ICQ functionality (communicating with the ICQ network and other ICQ clients) and it sends messages back to the user.

The server application is a threaded application (using the *POSIX threads* (pthreads) library. This provides a "private" connection to the ICQ network for each user.

4 Product requirements

4.1 Client side

Connection to ICQ using nothing but a web browser. This is the very heart of our system. A user can use any Java enabled browser on any machine connected to the internet in order to connect to their ICQ account and use all the basic features presented by the ICQ network.

Supporting basic ICQ features through a simple GUI. Our client supports all the basic ICQ features: sending and receiving messages and maintaining a contact list.

Working within applet security permissions. It is very important that the applet we produce does not conflict with the applet security permissions so it is indeed runnable on any machine that supports Java, even to the most strict extent.

4.2 Server side

Handle main load of communication with the ICQ server. Our client applet is designated to be as light at functionality as possible. The whole functionality of the ICQ client but the user interface is implemented in the server. This means that our server handles communication with the ICQ server for all active users.

Maintain contact lists for users. The server maintains a database of contact lists for all users. Therefore, no matter from which machine you connect, you will still have your very own contact list available.

4.3 Protocol

Provide flexible, easy to use interface to the system. We want the protocol to be both easy to implement to simplify the creation of other clients for our system and flexible enough to support new features if we chose to introduce them (for example porting another instant messaging network to the application).

5 Description of the problem — Software requirements

5.1 Client side

The client GUI is an applet that provides the following functionality.

Present GUI to user. The functionality of the client (and the entire ICQ network) is available to the user via a simple GUI for sending and receiving messages, viewing the contact list and supporting the functionality.

Connect to server using TCP/IP. The Applet maintains connection with a running thread of the server that is responsible for maintaining the ICQ functionality for that specific client.

Provide ICQ functionality. This includes:

1. Login to the ICQ network.
2. Show ICQ contact list.
3. Add user (ICQ UIN) to contact list.
4. Remove user from contact list.
5. Send message to an ICQ user.
6. Read received messages.
7. Logout from ICQ network.

5.2 Server side

Maintain threaded connection with applet to receive user requests. For each connected client there is a thread of the server (namely a pthread) that listens on its commands on one hand and passes them on to the ICQ network on one hand, and receives events from the ICQ network and forwards them to the client on the other hand.

Maintain connection with ICQ servers through ICQ protocol. The ICQ protocol requires certain operations to ensure correct functionality such as maintaining "keepalive" connection with the server, checking

for messages and more. The server should take care of these operations to make the client as light as possible.

Maintain and manage a database for contact lists. For each user there is an entry in the database that includes that user's contact list. The server is responsible for both creating the database entries and managing it, meaning saving the changes in the contact list to the database when necessary.

5.3 Protocol

The server and the client "talk" to each other using the standard TCP/IP connection provided by the POSIX socket interface (that is supported both in Java and in other POSIX compatible languages such as C/C++).

Over this link we implemented our own simplified protocol used for communication between the client and the server.

6 Competitive analysis — design alternatives

Here we will survey a few alternative unofficial ICQ clients and discuss the approach they have taken. Afterwards we will explain in greater detail the advantages of our approach and the problems of implementing it.

6.1 Survey on non-standard ICQ clients

Since ICQ is so widely used, the necessity of running a client on almost any working environment has been around for quite some time now. Since *icq.com* had been providing (at least at its first days) only one version for a single environment (Microsoft Windows 95), many unofficial and non-standard clients have been written, each with advantages and disadvantages relative to the official client and to our solution.

ICQ clients for Unices. [3] These are usually written in C or C++ and provide ICQ clients for Unix based systems (especially Linux). They provide the same functionality as the official client so we just mention them here for integrity of this section. We should also comment that

[3] Specifically we should mention *kicq*, *kxicq*, *licq* and *Micq*

such clients are the source of code libraries that provide open source implementation of the ICQ protocol.

Java ICQ. This client is an official release of *icq.com*. We put it in the "non-standard" section because unlike the regular client (written in MFC), it is written in Java. The main advantage of this is being able to run the application on any Java enabled environment. However, running a Java application is no different from running a regular application in terms of CPU consumption, security and anonymity of the user and needing to install the application on every machine that needs to use it.

ICQNet.com This is an implementation of ICQ in the form of a Java Applet that implements the ICQ functionality itself (i.e. contacts other users and the ICQ server independently). The advantage here is very clear — you can run ICQ anywhere without installing any further software. The main disadvantage is security. This applet violates the applet security permissions by contacting servers other than the one it has been sent from. Some Java machines will thus refuse to run this applet or display a warning for the user. In addition, the applet supplies the whole ICQ functionality by itself (unlike our implementation where the majority of the implementation is done by the server) which causes the applet to be very heavy.

Important note: We have been writing this document since December 2000. In the meantime, ICQNet.com and its partner webicq.com have stopped providing service. We suspect these companies have gone bankrupt, making a client like csicq even *more* necessary.

EBJava. This Java Applet based application is aimed at enabling users to manage a joint contact list for both ICQ and AOL IM.[4] The basic concept is very similar to our idea but the implementation is very weak: it is not fully operational, it uses some version of the ICQ protocol we are unaware of and it makes a very bad use of its database.

[4] America On Line Instant Mess-anger.

6.2 Advantages of our approach

Portability. The user needs nothing but a Java enabled web browser to use the product.

Light weight. The applet implements no functionality but user interface and connection to server, which makes it very light and easy to use on bandwidth limited platforms.

Privacy. All communication with the "outside world" is being done by the server, which makes the user completely anonymous (in terms of revealing one's IP address).

Easy to extend. Further functionality can be added to the system quite easily simply by adding more commands to the communication protocol. A change in platform (say to implement a client for AOL IM) can also be performed easily given the right library, as the client program is not dependent on the outside protocol.

6.3 Problems in implementing our approach

Lack of information. We rely on an outside library called *icqlib* to support connection between our server and the ICQ server. This is an open source project that was originally aimed at being used as a core of a graphic client (*kicq*) and as such has **no documentation**. In order to implement the project, we had to explore the *icqlib* package ourselves and we ended up writing our very own documentation for *icqlib* which is, for the best of our knowledge, the only public source of information about this package.

Need for non-standard solutions to integrate outside sources. We have an open source C library that should enable some interface to the ICQ network. In order to operate it we needed to "make it talk" with the Java applet. This problem has been partially resolved when we chose to implement the server in C and reduced to the problem of finding the suitable connection between a Java applet and a C server.

7 Top Level Design

The system will consist of the following parts:

1. Client — an Applet.
2. Web Server (Apache).
3. Csicq server (written in C).
4. A library supporting ICQ operations (icqlib).
5. A data Base for maintaining lists of authorized users and their contact lists.

These elements have been chosen to enable simple TCP connectivity, quick development and fair server load.

Following is a description of the elements which will be built for the system.

7.1 Client

The client program is a single Java Applet. The Applet will maintain a two-way TCP connection to the server. The Applet will exchange different type of messages with the server.

Incoming messages will inform the client of ICQ connectivity status, messages from other users and updates on members of the contact list.

Outgoing messages to the server will be sent as reactions to certain applet events: add member to contact list, send message, exit applet.

The major frames (i.e. parts of the window which comprises the Applet) of the Applet, used for user interface, will be:

Login frame. This will be active until user logged in successfully to the server.

List of the members of the user's contact list. This will be empty when the user starts the applet. The user can then add members to the list using another frame. The list will also show which of the members are currently online.

Received messages log. All messages received from other ICQ members will be printed in this frame.

Compose message box. This will be used to write messages to be sent to other ICQ members.

7.2 Server

The server application is a C program that is composed of two basic functional units:

Listening application. This is the main thread of the program. The listening application will listen on the csicq port and accept connections from users. Once a connection has been established, it is being managed by a new thread running the **connection control application**.

Connection control application. Every user has a thread in the server "taking care" of their connection. The connection control is responsible for both listening on the user port and forwarding the requests to the ICQ network and listening on the network, forwarding events from the ICQ network to the user.

The server uses three types of external mechanisms in order to enable its correct operation.

7.2.1 Database

The database can be either external or internal. Since porting an external database would force some structure on the code, we have chosen to use a flat database at current level.

The database is implemented as a flat directory in which there is a file for each contact list stored.

The format of the stored contact list is very simple. Each file contains pairs of lines. The first line contains the contact's UIN and the second contains its nickname. Therefore it is very easy to access the database in the C code.

We are aware of the flaws of flat database and therefore the functions that handle the database in the server are designed so that replacing the flat database with a "real" database would require minimal changes of code.

7.2.2 Threading Library

We use the *POSIX threads* (pthreads) library in order to enable threading of the application.

Being a standard in new POSIX compliant systems, pthread has been very extensively tested and is being used as a standard for threaded programming in big corporations such as IBM.

7.2.3 ICQ Library

The ICQ library that has been used is the publicly available *icqlib*. Icqlib is a library, implemented in C, which contains functions that provide multi-client API for the ICQ network and functionality.

The library is ported into the C code via linkage.

For more information, please refer to the icqlib manual that is provided with this documentation.

7.3 Protocol

The protocol wil be implemented over the regular socket interface. Each message is a string with a predefined structured that is being sent along the TCP connection.

8 Detailed design — implemented design and alternatives

At the basic level, the design of the project includes two components:

1. A Java applet that runs on the client side.
2. A server written in C that runs on the server side and communicates with the applets and the ICQ network.

This design has been chosen in order to accomplish the goal of the project: having a distributed ICQ client that would provide the easiest interface to the user.

The client has been written as a Java applet since this is an application that can run on almost any browser (or any other Java VM) and therefore would enable great portability for one's ICQ account.

The server, however, needed to handle great loads of communication (supporting multiple sessions at the same time) and therefore had to be written at the lowest possible level.

Since the server is required to send information to the user when an event occurs in the ICQ connection, we needed to have a bi-directional connection between the client and the server. This is why we chose using the socket interface to do that instead of alternative interfaces such as the Java object serialization.

Had we used object serialization, we would have a connection that could be initiated by one side only (the client). Therefore we would have to have some polling mechanism in the client side to poll the server from time to time and check for new messages that would be sent in return. This method is both bandwidth consuming and awkward to implement. The socket interface seemed to be a much better choice for our purposes.

The fact that we needed a low-level implementation and the fact that we could not use Java object serialization made the choice of development environment for the server very easy. A server written in C will be both light-weight and native to the socket interface.

In addition, the icqlib library that we use to connect to the ICQ network is native to C in system V environment (such as Linux). So using Java on the server side would not only require excessive tools and complication but also would not serve the purpose of making the application multi-platform.

The use of C as a main programming environment with pthreads and icqlib as library additions seems to be the best tool for the job.

8.1 Licensing

Our project is being released under the GNU-GPL license. This means that the source code is available to the public ("open source") and anyone can make changes in the code so long as they continue to distribute it under the GNU-GPL license.

There are various reasons for choosing this license:

1. We want other people to work on this project as well. We designed the application to be very flexible and easy to extend so if people continue working on that, they can easily add many useful new features.

2. Our code uses that icqlib library which is an open source application.

3. Opening the source enabled us to get a development environment at sourceforge[5] which provides many services for developers.

8.2 Client

The following is a general description of the CSICQ Client design and coding. Further information can be found in the API Documentation produced using JavaDoc.

The CSICQ Client enables users to connect and interface with the CSICQ server, which in turn connects them to the ICQ network. The Client provides Graphic User Interface (GUI) for the user. The GUI should be familiar to those who use the official Mirabilis ICQ client.

The Client is implemented using a Java Swing Applet and is capable of executing on any browser that supports the Java 1.2 API. The CSICQ Client main class is an extension of the Java Applet, JApplet.

The Client functionality is comprised of the following parts:

- *GUI*: (Graphic User Interface).
- *State*: describes the current behavior and changes of behavior.
- *Server Connection*: represents a message oriented connection to the CSICQ server.
- *Listening thread.*

8.2.1 GUI

You can see a screen-shot of the applet in figure 1.
The different GUI parts are as follows:

1. *Contact List*
   A list of ICQ network users. Each user-name is color-coded for their current ICQ network status:
   - Blue - user is on-line and available.
   - Red - offline or not available.

[5] http://www.sourceforge.net

A Click of a mouse on one of the user-names produces that user's UIN in two fields:

- The User Identification Number (UIN) field of the "Manage Contact List" Panel.
- The UIN field of the "Send Message" Panel.

2. *Login Frame*
   Enables the user to insert their details in order to log in to the ICQ network.

   Includes the following GUI elements: three text fields: UIN, nick-name and ICQ password and a button titled "login" which starts a login process.

3. *Manage Contact List Frame*
   Enables the User to add or remove users from their Contact List.

   Contains the following GUI elements: two text fields: UIN and nick-name fields, and two buttons: Add and Remove.

4. *Send Messages Frame*
   Enables the user to send messages (typed in the "Compose Message" text area) to other ICQ users.

   Contains the following GUI elements: a text field: UIN and a "Send Message" Button.

5. *Action Select Frame*
   Enables the user to switch between "Send Message" and "Manage Contact List" frames as well as to disconnect from the CSICQ Server.

   Contains the following GUI elements: three buttons: "Edit Contact List", "Send Message" and "Disconnect".

6. *Compose Message Text Area*
   Enables the user to type text messages they want to send to other users. this text area is cleared when the message is sent.

7. *Status Text Area*
   Shows all incoming and outgoing messages as well as other notifications.

Most of these GUI elements have a specific Event Listener Objects attached.

Those Event Listeners are invoked upon a GUI event (e.g. a button pressed), the action taken upon detecting such event depends on the current Applet state.

Almost all of these GUI elements are being inherited directly from a Swing Component (e.g. JButton, JTextArea). An exception to this "rule" is the Contact List, which is a combination of a GUI class and a Data Structure.

8.2.2 State

State defines the current behavior and functionality of the Applet. State changes according to local user actions and according to remote events or messages received. the State class controls the behavior of the following buttons:

- Login Button.
- Send Message Button.
- Add Contact.
- Remove Contact.

A typical state has the following structure:

- Constructor
  Send/Receives some message.
  Starts/Stops the listening thread.
  Disables/Enables Controls.

- Method Overrides
  The different methods responsible for events to reflect correct behavior (e.g. in the functional state the Send Message Button Event Handler will actually send the message to the CSICQ server).

State Descriptions:

1. *Initialized:*
   This is the initial state of the applet after being loaded. At this point, the applets presents all GUI elements disabled and tries to connect to CSICQ server.

   If the connection has been established correctly, the state is changed to login. In case an error occured (could not get socket or failed to connect to the server) the state is changed to Disconnected.

2. *login:*
   in this state, the login frame is activated and the user is prompted to enter their ICQ UIN, nickname and password. After the user has pressed login button, the applet sends a login request to the server and waits for the server to respond. If a "login successful" message is received from server, the applet state is changed to Get Contact List. Three unsuccessful login attempts change the state of the applet to Disconnected.

3. *Functional:*
   This is the fully functional state of the applet. In this state, all GUI objects are activated. They function as described in the GUI section. A thread is activated to listen to incoming events from the server.

4. *Temporary Disconnected:*
   This state is entered upon receiving an "Inform Disconnected" message from the server. The listening thread is not stopped. All GUI elements are disabled . The Applet is waiting for a "Login Successful" message and then it returns to the Functional state.

5. *Disconnected:*
   This state is entered upon a failure in the TCP socket connected to the server or upon closure of the applet. The listening thread is stopped (if activated), all GUI elements are disabled and all resources are freed. The Applet is ready to be closed.

8.2.3 Server Connection

Connection to the CSICQ server is handled by the ServerConnection class. The Server Connection is a message oriented network connection to the server. The Server Connection is responsible for creating the TCP socket

to the CSICQ server as well as sending and receiving messages from the CSICQ server.

It attaches two classes: MessageInputStream and MessageOutputStream to the raw TCP Stream. These two stream classes, along with the Message classes, enable a modular and convenient transportation of CSICQ protocol messages to and from the server.

8.2.4 Listening Thread

This is the main thread except for the Event Thread (the Event Thread is the only thread where Swing Events take place).

The Listening Thread, as it's name suggests, Listens on the Connection object for incoming messages from the CSICQ server. Each message received is first identified for it's type, then an action is taken. The actions taken by this thread are usually considered with the GUI elements of the applet.

The following Event messages are currently dealt with in the listening thread:

- Received Message: a message from an ICQ network user, the message is printed on the status text area.
- Update Contact: an update of the status of a contact list member. The Contact List object is updated.
- Push contact: the CSICQ server is adding a member to the Contact List from the database. The member is added to the contact list.
- Inform Disconnected: a notification of a disconnection from the ICQ network. The applet state is changed to Temporary Disconnection.
- Authorize login: the server has successfully logged the user into the ICQ network. the applet status is switched to Functional.

Since the GUI elements of the Applet are manipulated in the Event Thread and can happen at the same time, some kind of synchronization method is being used.

The method of synchronization implemented is using the SwingUtilities Invoke Later method. This method simply schedules a runnable object for later execution on the Event Thread, thus all GUI changes are performed in one thread with no hazards.

8.3 Server

The csicq server is an application written in C and compiled in Linux.

The main goals of the server are as follows:

1. Listen on the predefined designated icqlib port and accept new connections.
2. Provide a separate thread for each connection.
3. Receive requests from the user and forward them to the ICQ network using icqlib.
4. Receive events from the ICQ network using icqlib and forward them to the user.
5. Maintain and manage a database for users' contact lists.

We will now describe in detail how the server accomplishes these goals.

8.3.1 The listener

This is the application that is responsible for listening on the csicq socket and accepting the connection. It is implemented in the files:

- listener.h
- listener.c

The functionality and implementation of the listener are very simple. The procedure contains a standard TCP socket that is set to listen on the csicq port (that has been set to 4040).

We use `select()` to check for incoming connections and when one arrives, the listener simply `accept()`s it and provides it as argument to the connection control function. The connection control function is being activated in a separate thread created by the `pthread_create()` function. After the new thread has been created, the listening application goes on listening.

8.3.2 Connection control

This application runs on a separate thread for each connected user. It is responsible for listening on the user socket on one hand and listening on the ICQ network on the other. The connection control function is implemented in the files:

- connector.h
- connector.c

In the connection control module we are introduced to the structure of "link environment" (`struct link_env`). This structure includes the `ICQLINK` of the connection (which is the session identifiers for the icqlib library as can be read in the icqlib manual submitted with this documentation) along with additional information about the connection. The link environment contains the following properties:

- The `ICQLINK` of the connection.
- An identifier to whether the contact list has been initialized.
- The user's contact list. The contact list is implemented as a linked list and will be described in detail when we talk about the database.
- An identifier to whether the link is active.
- A pointer that can point to any additional data.
- The socket of the connection to the client.
- An identifier to whether the session has ended or not. This is required for purposes of flow of control.

The main function of the connection control mechanism is `connector()`. This function is called when the thread is initiated and handles flow of control until the connection is closed.

The function first initializes the environment and then goes on a look that is active as long as the identifier in the environment is set to a true value.

In the loop, we check for information from the ICQ network and for information from the user.

- If the connection to the ICQ network has been established (there is an identifier for that in the environment), the function first checks whether it's time to send another keepalive message to the ICQ network. If so, it sends a keepalive message. Then it checks for new messages from the ICQ network using the `icq_Main()` function. This function calls the appropriate callbacks if necessary (please check the icqlib manual for more information on the structure of icqlib).

- The function then checks for new information on the client connection and if there's a new information on that, it extracts the message and handles it. Extracting the message is simply a matter of socket programming. Handling the message is also made simple by our protocol structure. All we have to do is read the message type and send the message to the appropriate handler (we will elaborate on the handlers later).

Before the connection control function returns, it cleans the environment, releasing all allocated resources.

8.3.3 Handlers

The handlers are functions that are responsible for specific operations in the application. A handler could be invoked as a result of an event in the ICQ network or as a result of a user request. The handlers are implemented in the files:

- handlers.h

- handlers.c

The description above gives a very intuitive separation of handlers into two basic groups:

ICQ handlers are handlers (also referred to as "callbacks") invoked by events in the ICQ network. In fact, they are being called by the `icq_Main()` function. Please refer to the icqlib manual for more information on callbacks.

The convention in the code is to name these handlers `icq_handle_`x where x is the event to be handled.

The callbacks implemented in the current version of csicq are:

- `icq_handle_logged_in` which handles connection establishment messages from the ICQ network, informing the client that they are connected to the network and initializing the contact list.
- `icq_handle_not_logged_in` which handles messages from the ICQ network notifying that we could not login to the network, notifying the user that the attempt was unsuccessful.
- `icq_handle_disconnect` which handles messages from the ICQ network notifying that we have been disconnected from the network, notifying the user of that and trying to reconnect.
- `icq_handle_recv_message` which handles the event of a new message received by the user. The handler composes a message to the user using our protocol and sends if over the TCP connection, notifying the user of the incoming message.
- `icq_handle_update_status` handles the event of a change in status of one of the user's contacts. The handler composes a message to the user using our protocol and sends if over the TCP connection, notifying the user of the change in status.
- `icq_handle_user_online` handles the event of a contact coming online. Similar to `icq_handle_update_status`.
- `icq_handle_user_offline` handles the event of a contact going offline. Similar to `icq_handle_update_status`.
- `icq_handle_arv_ack` handles the event of receiving server ack message. This handler is not used by the current version of csicq and used for testing purposes only.

User handlers are invoked as a response to an event on the user connection. The user sends a message using our protocol, it is being received and then interpreted by the `handle_message` function (which is not really a handler but rather a dispatcher of messages to handlers). The function determines according to the message type which user handler should be invoked and activates it. The naming convention for these handlers is `handle_`x where x is the event to be handled.

The handlers implemented in the current version of csicq are:

- `handle_login_req` handles a login request fro mthe user. It initializes the `ICQLINK` of the connection and attempts to connect to the ICQ network.

- `handle_disconnect` handles a disconnection request from the user. Namely it ends the session.
- `handle_send_msg` handles a request by the user to send a message. The handler calls the appropriate icqlib function to forward the message to the ICQ network.
- `handle_add_usr` handles a user request to add some contact to their contact list. The handler adds the new contact to the list of contacts and notifies the ICQ network of the addition.
- `handle_del_usr` handles a user request to remove a contact from their contact list. The handler removes the contact from the contact list.

8.3.4 Service routines

This is a group of small auxiliary routines that are frequently used by other modules. They are implemented in the files:

- services.h
- services.c

These function include:

- `ready_to_read` tells when a socket can be read.
- `receive_message` extracts a full protocol message from the socket.
- `clean_buf` frees buffers allocated by `receive_messsage`.
- `message_type` extracts a message type from a message.
- `ready_to_write` checks whether a socket is ready to be written into.
- `send_message` sends a full message over a socket.
- `announce_connection` prints the information of a new connection to the log file.

8.3.5 Contact list and database structure

In order to implement the contact list, we implemented a *linked list* which is a well known data structure. Our linked list has two types of data entries — a UIN and contact name. The linked list is implemented in the files:

- contactlist.h
- contactlist.c

The operations over the linked list are rather standard and include adding a new entry to the list, searching for a specific entry, deleting an entry from the list and cleaning up the list. We also added special features that enable reading a list given a file descriptor. This includes locking the file (to avoid two threads trying to write to a file at the same time) and reading the contacts from the file.

Then we implemented a flat database that uses the linked list. The database knows the naming conventions of the database, it opens the correct file. Then it uses the linked list functionality to read or write the contact list. The flat database is implemented in the files:

- db.h
- db.c

Then we implemented a set of high-level functions that can handle a link environment and call the correct database functions in order to initialize the contact list, add or remove contacts and terminate the contact list when needed. These functions are implemented in the files:

- contacts.h
- contacts.c

8.3.6 Database conventions

The conventions of the database are as follows:

1. The path of the database is indicated by the constant `DB_PATH` (see db.h).

2. The contact list on a certain UIN is stored in the database path in a file called *<uin>*.cl The extension is determined by the constant DB_EXT which is currently set to ".cl" (see db.h).

3. The structure of a contact list file is a sequence of two-row entries. The first row indicates the UIN of the contact where the second indicates its nickname (see contactlist.c).

4. A contact name's length is at most MAX_CNAME_SIZE characters (see contactlist.h).

5. A UIN cannot appear twice in the contact list.

8.3.7 Logging

The server supports logging inherently. If you want the server to print log messages, all you have to do is define the DO_LOG flag and set the LOG_FILE definition to influde a FILE * of the file you want to write the log into (for example setting LOG_FILE to stdout will print the log messages to the standard output).

8.4 Protocol

Each message in the protocol is a string composed of the following parts:

1. Total message size (including the size) - 4 bytes field (unsigned long). This is important in order to enable the receiver of the message to know when the current message ends.

2. Message type - 4 bytes field (unsigned long). The complete list of protocol messages is described below. Since every type of message has a structure of its own, the receiver needs to know which message they received.

3. Message body - according to the message type.

8.4.1 Message types

Our protocol is build in a way that is very easy to extend. This is before the message types are assigned in hierarchal order so you could very easily add a completely new class of messages to the protocol.

The convention of the messages is as follows:

Preliminaries: Predefined constants.

$100 + x$ means that the message is of type x and that it is a message from the client to the server.

$200 + x$ means that the message is of type x and that it is a message from the server to the client.

Messages from the client to the server: 100+

- 1 - Request for login. Body: uin, password, nick.
- 2 - Send message. Body: destination uin, message text.
- 3 - Add user to contact list. Body: uin, name.
- 4 - Remove user from contact list. Body: uin.
- 5 - Disconnect. Body: empty.

Messages from the server to the client: 200+

- 51 - Authorize login. Body: empty.
- 61 - Login refused. Body: empty.
- 2 - Received message. Body: from uin, text.
- 3 - Update user status. Body: uin, new status.
- 4 - Inform disconnected. Body: empty.
- 7 - Push contact. Body: uin, nick.

8.4.2 Message field implementation

In the body of the message there are two types of fields, one type is UIN which is implemented as a regular unsigned long number, the second type is text fields. These include name fields and passwords.

For the text fields we needed to come up with a new structure since if we just sent the string, we wouldn't know where the string ends and the next field begins.

Therefore a text field is composed to two sub-fields:

1. String size - the total size of the null-terminated string (including the NULL). This is a 4 bytes field (unsigned long).

2. Text - this is a variable length null-terminated string (char[]).

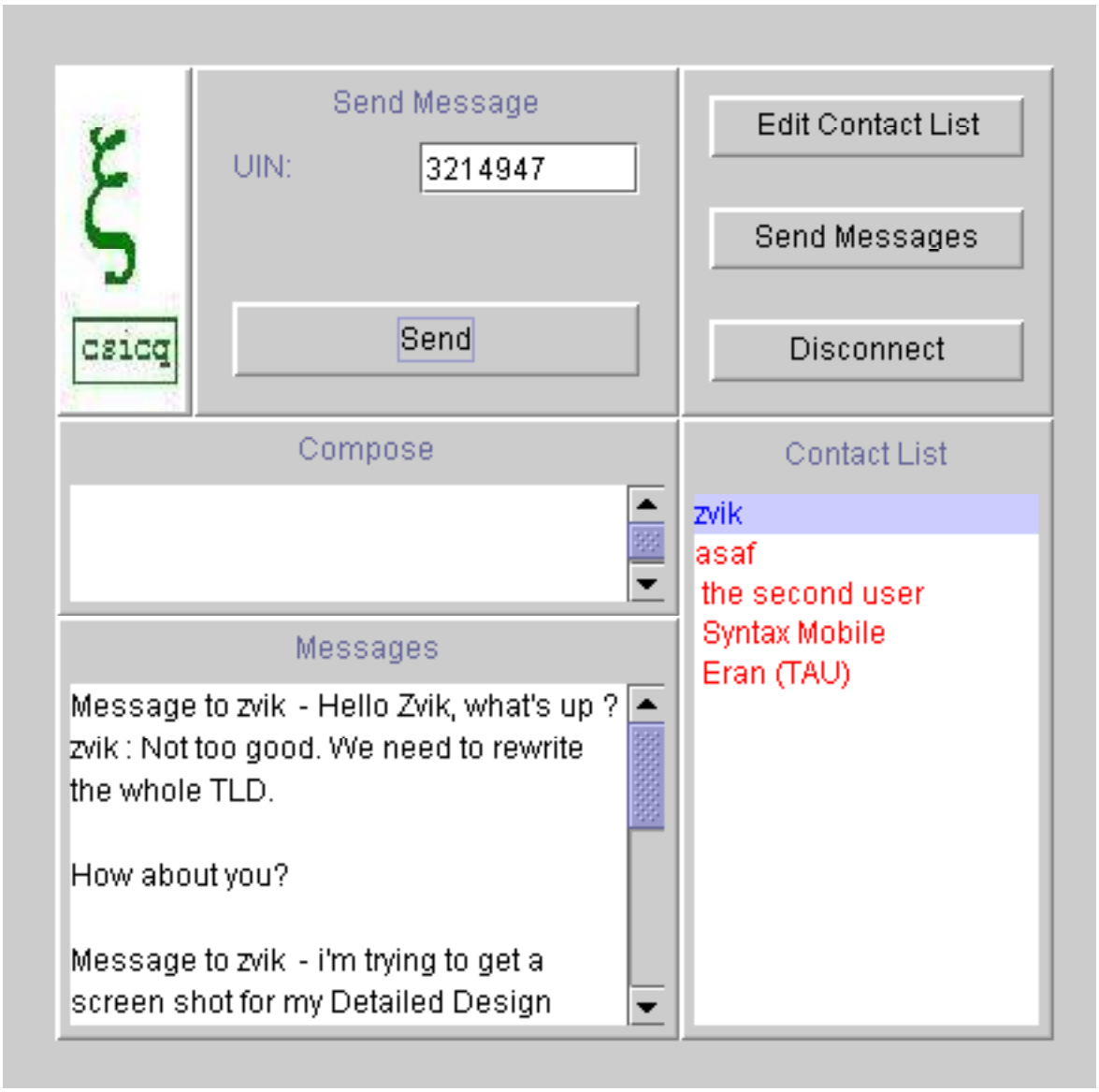

Figure 1: A screen-shot of the applet