Client-Server design

The messenger client and server will be implemented using Java sockets. All messages passed between client and server will be plain text, with the body of the message itself being encrypted by the client before transmission. The server will contain a relational database that will, at a minimum, contain the following information:

Username
Passphrase
User’s public key
Basic user info (last IP address, email address, etc.)
Status (online, away, etc.)
Associates (‘Buddy list’)

Additional info will be required depending on the plug-ins added (i.e. accepts file transfers, etc.)

The database will preferably be Oracle or mySQL, or a similar relational database, accessible through a JDBC interface. This will first be run on a Linux or Windows platform.

Various messages will be defined for communication between client and server, and at minimum will include the following:

(Client->Server)
REGISTER – creates a new entry for the user in the database. Information required would be a username, the user’s public key, and an email address (along with any other needed information) to establish the user’s account.

LOGIN – the client informs the server it’s online by encrypting a passphrase with its private key and sending it to the server with its username. The server would then change the user’s status to online.

ADDBUDDY – the server adds a buddy to a client’s list of associates, and returns the buddy’s public key to the client.

NUKEBUDDY – the server removes a buddy from the associates list.

NEWMESSAGE – sends and encrypted message to be delivered to another specific user. This is particularly useful since some firewalls won’t accept incoming connections, but this way the server can send data back through the hole the client already punched.

(Server->Client)
YOUALIVE? – checks to see if a client is still online (or should just be able to tell if the socket is still open or not…)
NEWKEY – send the new public key of one of the client’s associates.

NEWPASS – sends a new passphrase (encrypted with the client’s public key) to be used for logging in.

BUDDYON/OFF – one of the client’s associates has changed status.

Various other messages will need to be defined to initiate the various dialogs for transferring a file. Other flags will be included as leading / trailing bytes to indicate beginning or ending of the message as well as different states (ie: encrypted or plain text)

**Encryption Scheme**

The encryption scheme was originally planned for very high number of bits (around 16K bits), but this does not make it feasible to transfer files in encrypted mode. Instead of using two separate keys, we plan on using one key of roughly 1KB – providing sufficiently secure communications considering today’s computer power. Given enough time, we allow for variable key lengths.

The encryption algorithm is simply going to be based around the java bignum class, and public key cryptography. By using very large prime numbers we can guarantee the likelihood that the message will arrive intact and not understood by a third party.

One other option we are looking into for our encryption and communication channel is a new proposal for a protocol called SILC *(Secure Internet Live Conferencing)* which allows not just chat but other forms of communications. This decision will be made shortly.

**GUI**

The user interface will be where the fun really comes in. We plan on supporting all the common options such as control over fonts, backgrounds, emoticons and other bells and whistles given the time. We would also really like to be able to add in support for extensible modules that allow any Java developer to freely create plug-ins that piggyback their data on the chat conversation. One simple example of this would a checkers game that can be played while chatting with that same person. Other more practical uses of the technology would multimedia, allowing for phone calls or other voice data to be routed across the chat channel as well as video with the appropriate hardware.

The GUI will have a main application that lists options and the status of your “buddies” – whether they are online, away, offline, out to lunch, etc. We would like to be able to allow for the entire application to be customized so that you could have a custom icon loaded and displayed to each buddy – including emoticons.

The GUI will be the interface that sits on top of the other components and displays the information they provide to it as well as provide them with proper and sufficient data.