6. Design

6.1. High-Fidelity Prototype

The preliminary, low-fidelity prototype confirmed the technical feasibility of cross-lingual IM (see chapter 4 – Groundwork, pp. 59-64). The high-fidelity prototype, built over the foundations laid by its proof of concept counterpart, had two major purposes:

1. Explore the usability of a cross-lingual IM developed, following UCD principles, with input from its future users to match their needs and requirements as closely as possible.
2. Provide a realistic and familiar system for cross-lingual IM testing and evaluation, in a realistic environment and context of use.

The design was determined by the views, preferences and expectations of the target user group (see chapter 5 – Informing Design, pp. 65-100). Current standard IM features, such as video, drawing and file transfer, were not included for being beyond what is required for testing cross-linguistic communication.

6.2. Architecture Overview

The architecture was a typical client-server structure depicted on Figure 10 below.

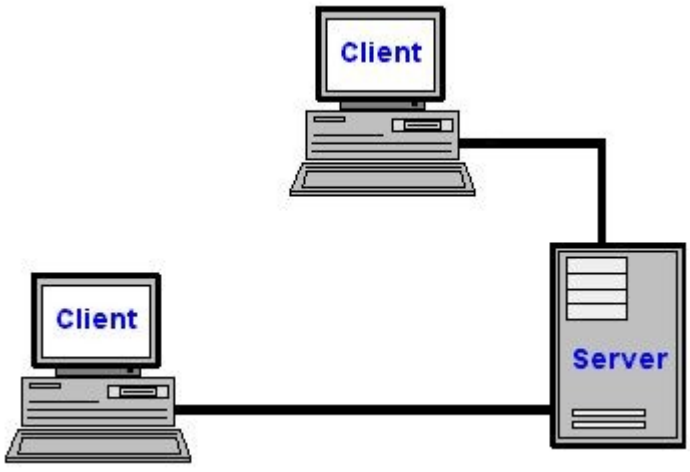

Figure 10 - The client-server architecture adopted

6.2.1. Client-Server Architecture

The client-server architecture was chosen because of its innate multi-user support, and its modular and message-based networked model. These characteristics are appropriate to the distributed and message-based nature of IM (Edelstein, 1994; Schussel, 1995). The server, provider of the IM service, mediates conversations and runs the business of managing users, logs, messages, languages, translations and the synchronization of all interactions. A client represents an IM user, sending and receiving requests for services to and from the server.

As refinements of their low-fidelity predecessors, both client and server modules were developed using the Python programming language (Rossum, 2001). Python was found to be productive and efficient in the development of the preliminary prototype (see chapter 4 – Groundwork, pp. 59-64). Therefore, a change of programming language was not required nor recommended due to the time constraints. Instead of implementing one of the existing IM protocols used by current IM systems, such as SIMPLE[1], OSCAR[2], or XMPP[3], a custom and highly simplified protocol for communication between client and server was developed. This alternative was preferred because existing protocols are verbose and involve manipulating data beyond the scope of this project. A protocol like OSCAR, for example, contains additional information not handled by the IM systems that use it.

6.2.2. The Server Module

The server module is a refinement of its low-fidelity counterpart (see chapter 4 – Groundwork, pp. 59-64). An overview of its behaviour is illustrated by the flow chart of Figure 11.

1 Available at: http://www.ietf.org/html.charters/simple-charter.html

2 Available at: http://joust.kano.net/wiki/oscar/moin.cgi/InstantMessages

3 Available at: http://www.ietf.org/rfc/rfc3920.txt

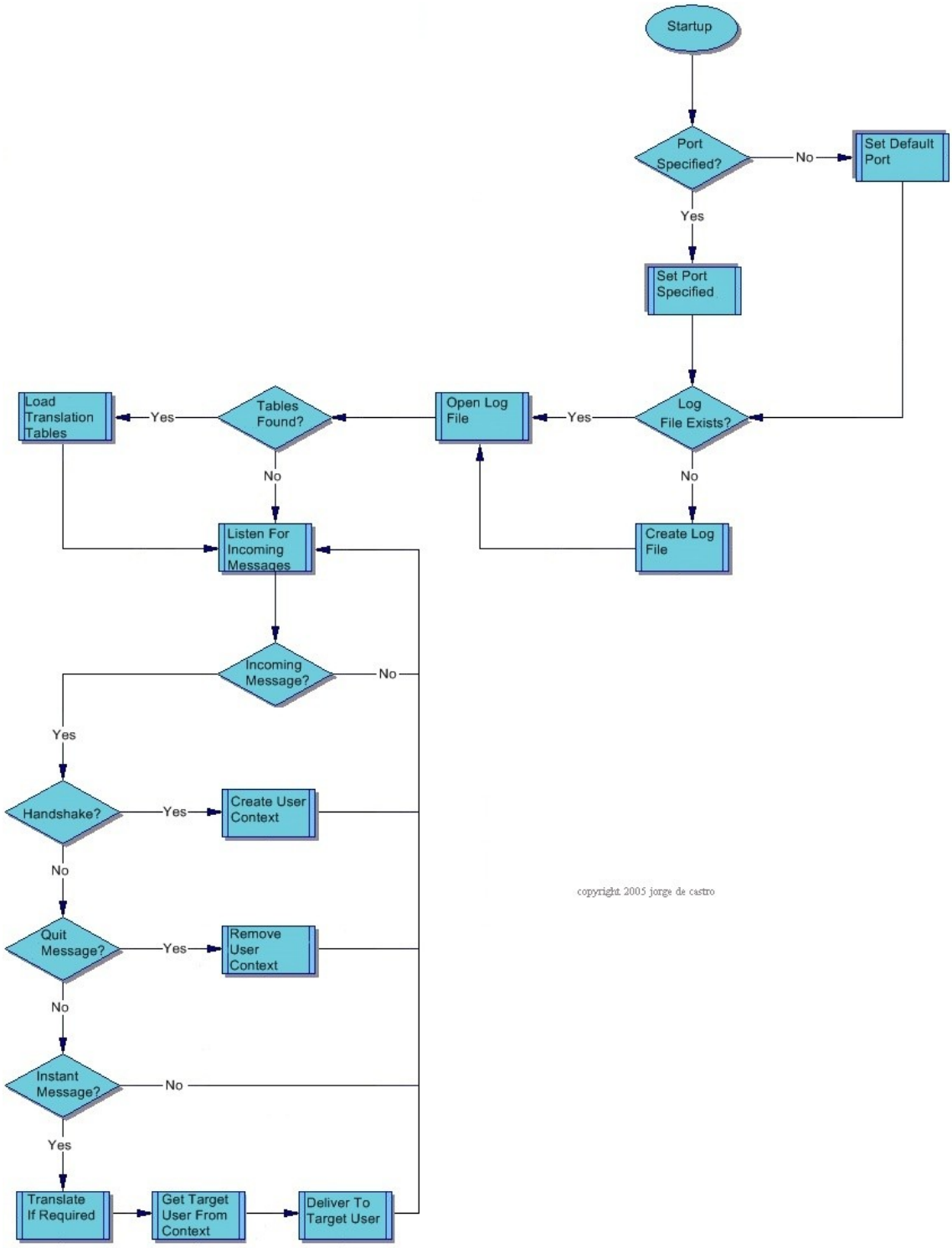

Figure 11 - Task flow chart for the server application

Sequence of Events

The server application can be terminated at any stage after start-up by typing *QUIT* on the server console. Exiting the server application releases all resources consumed and reserved. The flow chart of Figure 12 below illustrates the exit task. Since the *QUIT* command can be issued at every node of the flow chart of Figure 11, it is shown here separately for simplicity.

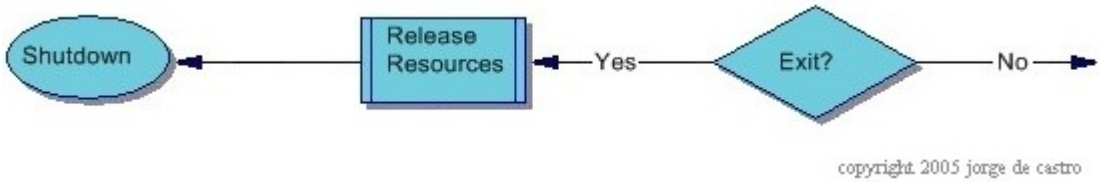

Figure 12 - Exit task flow chart for the server application

Until a shut-down command is received, the server listens to a designated port for incoming messages. The port is a channel specified at server start-up for communication with clients. If no port is given the default port number "123450" is used. Clients should assume this default port number unless instructed or specified otherwise.

If a port is available and a communication channel established, the server accesses a log file named *pyserver.log*. If the file does not exist it is created; otherwise it is opened for writing. This operation initiates the logging process that will archive on the server all instant messages exchanged between clients. The log file can be subsequently read with any text editing application, such as Notepad on Windows and VI on UNIX, to analyse the behaviour of the system and the transcripts of user interactions.

After setting-up the logging system, the server attempts to load from the file system any file containing translation of acronyms, abbreviations and alternate spellings between source and target languages. These files are automatically read by the server application and can be deployed on a per language-pair basis without the need to modify any server code. For this project, a file for the English-Chinese language pair was deployed, containing translations of abbreviations, acronyms, and alternate spellings from English to Chinese.

The first connection received from a client is a handshake message to authenticate the client and create an internal user context. This user context stores the client's desired nickname, list of buddies, language preferences and connection details on the server. The client is authenticated (and the user context created) if and only if both the chosen nickname is unique across the system, and the maximum number of allowed clients has not been reached. Authentication consists of sending a successful handshake-acknowledgement message back to the client.

After successful authentication (via the handshake procedure described above), any subsequent message from the same client causes the server to retrieve the sending user's nickname, preferred language and buddies from its internal user context. If the message received is a *QUIT* message from a client, the server removes the corresponding client from its user context and acknowledges receipt of the quit command to the exiting client. The client can then exit gracefully and its nickname and communication channel are made available to reuse. Otherwise, the server checks if the language preferences are set for the participants in the conversation, and if the language at the source differs from the language at the destination. If the source and destination languages differ, the instant message is passed on to the translation sub-system; otherwise, the message is delivered to the target user, who is extracted from the server's internal user context. The translation sub-system is simply an interface that delegates translation of messages to a dedicated MT system. In this project, the MT system chosen due to its free[4] availability was AltaVista's Systran -widely known as AltaVista's Babelfish[5]. Because Systran is an external system, each request for message translation is sent to the remote AltaVista service, which returns a Web-page containing the translated text back to the server. The server then parses the Web-page to extract the translated message. Both original and translated messages are then packed, using the custom communication protocol, and delivered to the target user. The round-trip to AltaVista's system adds approximately 2 seconds of delay on average to each interaction, but this latency can be avoided if the MT system is local to the server.

A screen-shot of the server module running on a Windows XP console is shown on Figure 13 below.

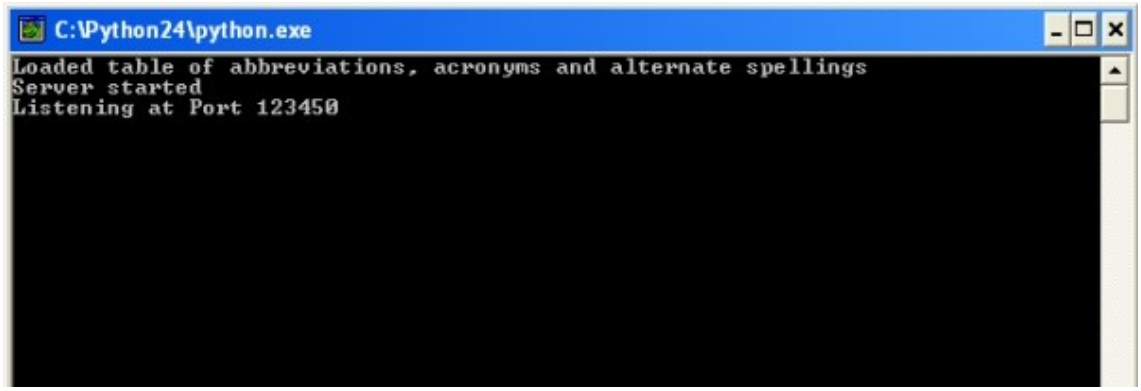

Figure 13 - Cross-Linguistic server prototype deployed at zooropa.nidelven-it.no

4 Free as in “Free Beer”

5 Available at: http://babelfish.AltaVista.com/

6.2.3. The Client Module

The client module is a refinement of its low-fidelity counterpart (see chapter 4 – Groundwork, pp. 59-64) with an added graphical user interface (GUI) with a design familiar with -and inspired by- that of existing IM client applications.

Upon start-up, the client's first screen requests and collects handshake information to send to the server. This includes the server URL, the port number, the nickname for the user and his language preferences. The port number should be the same one the server is listening to. If the default port number "123450" is changed at the server the client must adapt accordingly. For simplicity reasons, users are only required to provide their desired nickname without a password. When all the information is entered and the *Login* button pressed, the client application is ready to authenticate the user with the server. A picture of the client's first screen is shown in Figure 14 below.

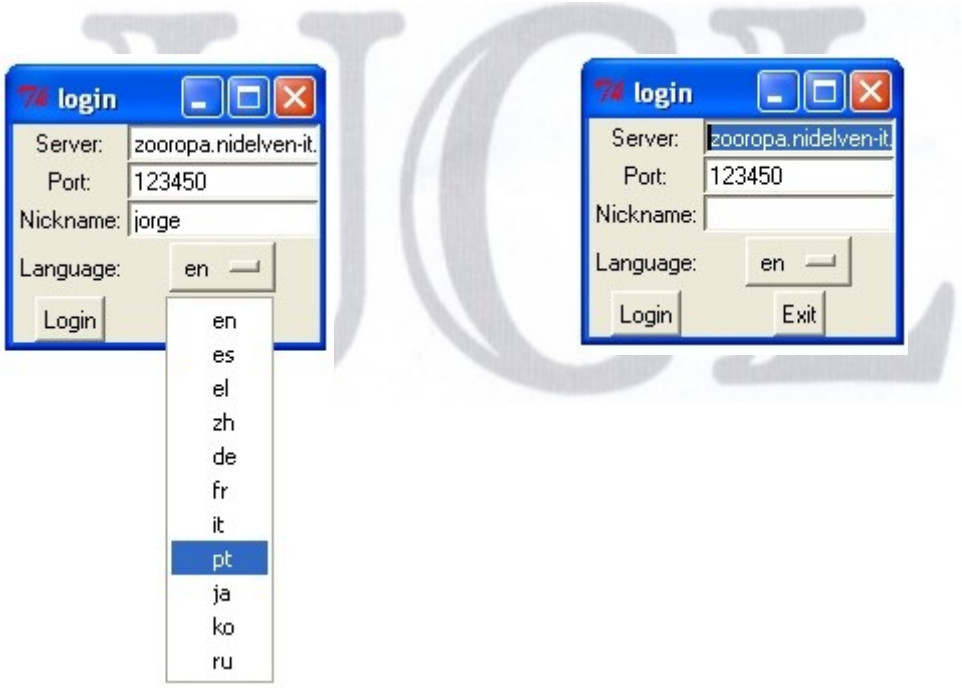

Figure 14 - Cross-Linguistic IM client configuration

If the nickname chosen is already in use by another user, the user is requested to choose a different one; if the maximum number of clients the server can handle has been reached the user is unable to proceed; if an unknown error occurs the user is also unable to proceed. In the event of any error the system informs the user via a pop-window and exits.

If no problems occur, the client registers with the server and the server indicates its acceptance by returning the authenticated nickname and preferred language back to the user. When a connection is established, the IM window of Figure 15 below replaces the login and configuration panel of Figure 14.

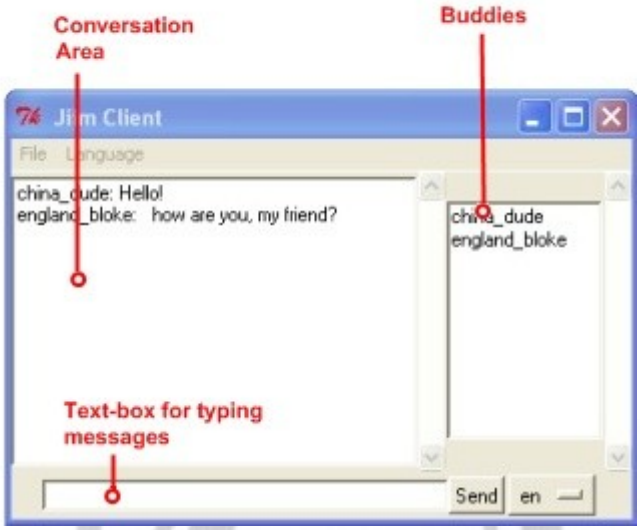

Figure 15 - Cross-lingual IM window

The user composes instant messages by typing them in the corresponding text-box and pressing the *Send* button, and reads incoming instant messages from his buddies in the conversation area of the window (see Figure 15 above). Incoming messages can be of the following types:

1. A presence indicator of a new buddy joining the conversation, in which case the client application stores and displays his nickname in the Buddies area of Figure 15 above.
2. An incoming message from a buddy participating in the conversation, in which case the client application displays his instant message in the Conversation area of Figure 15 above.
3. A *QUIT* acknowledgement message from the server, received after a *QUIT* message issued by the client, and in which case the client application exits gracefully releasing all its resources consumed and reserved.

Figure 16 below illustrates two clients (users) engaged in a cross-linguistic conversation.

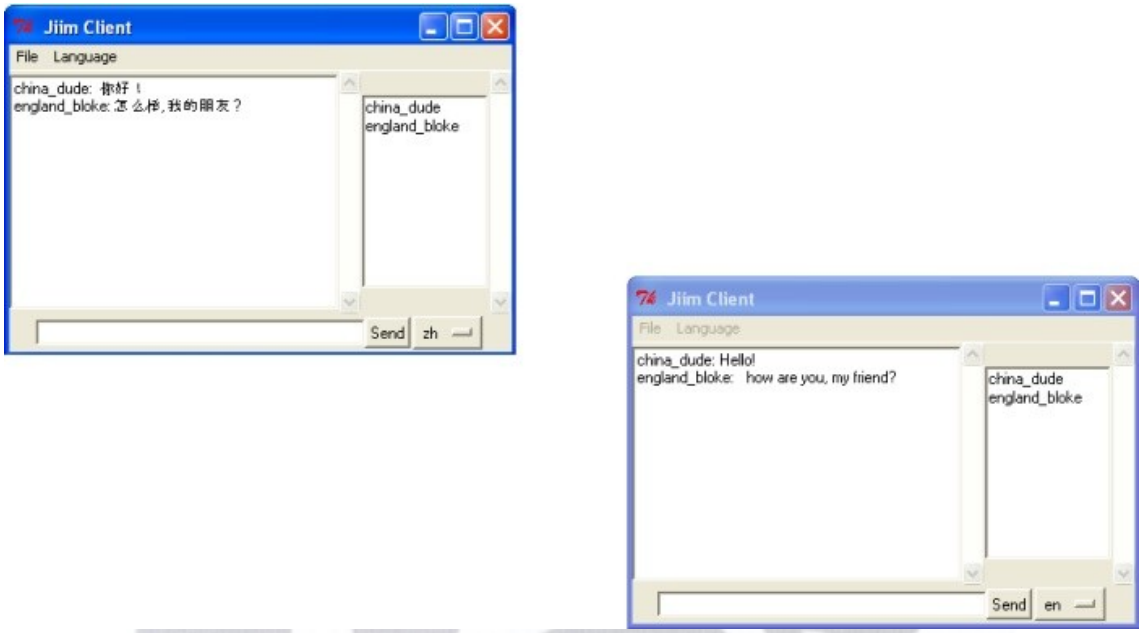

Figure 16 - English-Chinese interaction

Participant on the window to the left hand side is typing and reading in Chinese (his L1), and the participant to the right hand side window is typing and reading in English (his L1). The language to read instant messages is specified on the *Language* toolbar menu option at the top of both windows. The drop-down menu to the right hand side of the *Send* button displays the language specified by each user as his preferred to type instant messages. Participant on the left hand side window is typing in Chinese (ISO code "zh"), and participant on the right hand side is typing in English (ISO code "en"). Participants can change the language used to type instant messages by clicking on the drop-down menu and selecting a different language from the list.

6.3. Design Justifications

The design of the high-fidelity prototype was determined by the communication patterns, views, perceptions, opinions, feelings, experiences and expectations –observed, inferred and reported- of members of the target user group (see chapter 5 - Informing Design, pp. 65-100). The prototype's GUI was informed by the three information-eliciting methods employed, namely, survey, interview and observations, and log analysis. This triangulation, an investigation from three different angles with distinct methods, increased the reliability and validity of the analysis. The way the data collected through each separate method informed and determined the design and development of the prototype will be explained in the following sections.

6.3.1. Interview and Observation Data

The results from the interview and observation analysis highlighted, amongst other things, that users are parsimonious about their IM screen estate, and know where they want or expect cross-lingual interaction components, namely, the controls to set their language preferences, the display of their current language settings, and the display of the language preferences of their buddies (see chapter 5 – Informing Design, pp. 71-75). The choices were consistent across interviewees and their preferences were followed strictly on the design of the client's GUI. Figure 17 below displays the user expectations reported in the interview and observation sessions and the GUI of the high-fidelity prototype. There is a clear overlap between the component locations expected by users and those applied on the GUI.

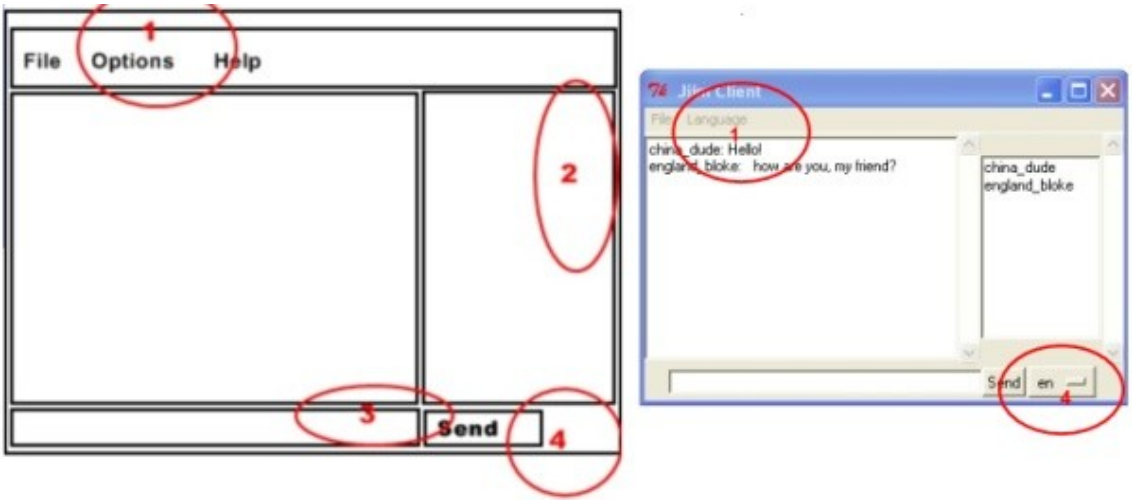

Figure 17 – Diagram with user expectations on left hand side, prototype's GUI on the right hand side

The configuration panel illustrated in Figure 14 was implemented as a separate window because interviewees expressed preference to hide details and procedures that do not change nor are accessed often.

Interviewees expected to specify their preferred language for reading incoming instant messages at location 1) in the diagram side of Figure 17. On the prototype's GUI, they can specify the preferred language to read instant messages at location 1) on the prototype side of Figure 17. Location 1) is also where they can alternate between original and translated text on demand (when enquired about how they wanted to handle original and translated messages users reported preference to see one text at a time). This option also satisfies their desire for IM as a language-learning tool, since users can confirm and learn on-demand the translation of unknown words. Moreover, interviewees reported spending time clarifying comments and explaining meaning of words, something that can be improved with the availability of translated text.

Interviewees wanted to specify the language for typing outgoing instant messages at locations 3) or 4) in the diagram side of Figure 17. Location 4) was chosen because more interviewees preferred it. The location 3) alternative could be explored further on a subsequent iteration. Interviewees expressed the desire to be able to quickly change the language setting on a per-conversation basis. This is because they might want to enable translation for some users and disable it for others, depending, for example, on their buddy's fluency in the language common to both. On the GUI, the typing language preference is shown in the drop-down menu to the left of the *Send* button (location 4) on the prototype picture of Figure 17. Users can quickly override the language setting on-demand by choosing another language from that drop-down list.

Location 2) on the diagram of Figure 17 specifies the language preference of the buddy -ignored in this high-fidelity iteration but likely to be included in the next.

The interview sessions provided a comprehensive list of essential functionality that users wanted and expected, together with where in the IM screen estate they expected the interaction components that gave them access to this functionality. The preliminary requirements and use cases (see chapter 4 – Groundwork, pp. 49-54) were rewritten and revised accordingly, incorporating the additions and preferences reported (see Appendix A, pp. 138-143).

6.3.2. Survey Data

The 162 survey responses obtained provided many statistically significant results (see chapter 5 – Informing Design, pp. 78-79). The fact that more than half of the population sample (54.76%) reported having conversations with non-native speakers of their language validates the need for some cross-lingual support. This support must be flexible and customisable on-demand, because not all conversations were cross-lingual (55.24% are not). Because preference to type in L1 was statistically significant and preference to read in L1 was not, the prototype was made to allow separate language configuration for reading and typing. It is likely that a user might want to type in his L1 and read incoming messages in L2. Thus, on the prototype, users were given the ability to enable or disable cross-lingual support via the drop-down menu to the left of the *Send* button and the *Language* toolbar option. This can be seen in the prototype side of Figure 17: configuration of reading preferences is done in location 1), and configuration of typing preferences is done in location 4).

The preference to type in L1 is strongly related to the percentage of emoticons, abbreviations, acronyms, and alternate spellings used. This fact indicates the MT system handles small subsets of each message -the original message minus emoticons, abbreviations, acronyms, and alternate spellings, thus improving the quality of the translated content.

Users can tell when their conversation buddy is not a native speaker of their language with statistical significance. This seems to indicate there would be little benefit from having the native tongue of their buddy clearly identified or accessible from his profile on the IM interface as initially intended (see chapter 4 – Groundwork, pp. 59-64), but this claim could be tested in the next iteration of the prototype.

6.3.3. Log Data

The log data consolidated the qualitative aspects of the interview and survey data with quantitative evidence from real systems in real settings and context of use, and confirmed quantitative aspects of the fieldwork methods employed (see chapter 5 – Informing Design, pp. 80-82). It also contrasted L1-L1 from L1-L2 interactions and informed the development of the translation sub-system.

The average of 5.75 words per turn found showed messages exchanged are generally short in length. This fact combined with the large number (of approximately 50%) of emoticons, abbreviations, acronyms, and alternate spellings found per message simplified the translation task, implying higher quality for the translation outcome. Translation was developed as a multi-step operation: in the first step, emoticons, abbreviations, acronyms, and alternate spellings from the source language are translated into their equivalent in the target language, independently of the dedicated MT. The accuracy and quality of the message content was improved because typical English IM jargon, such as “brb” (“be right back”) and “cul8r” (“see you later”), is translated into their exact equivalent in Chinese; In the second step, the remaining text is translated via a dedicated MT.

Moreover, the relatively small vocabulary found across languages indicated ways to optimise certain known or frequently observed combinations of words or sentences in the same way that abbreviations, acronyms, and alternate spellings are handled. This could be included in the next development iteration.

6.4. Benefits

The major advantage of having target users inform the design through fieldwork and other HCI methods of data collection comes from obtaining a final product that supports their needs and matches their expectations (Wiklund, 1994). However, this is a contribution measured in terms of the final design outcome, thereby overlooking the multiple contributions that were gained throughout the course of the development process itself. Hence, during the iterative development of the prototype, the UCD process employed brought the following direct benefits:

1. Reduced number of development errors due to the up-front provision of large amounts of relevant information regarding user needs and expectations (see chapter 5 – Informing Design, pp. 65-82). The information gathered eliminated the need for trial-and-error- based development.
2. Reduced number of iteration runs due to the up-front provision of large amounts of relevant information regarding user needs and expectations (see chapter 5 – Informing Design, pp. 65-82). The information gathered allowed right decisions quicker and correct actions taken in fewer steps.
3. Identified sooner rather then later specific requirements and functionality that needed to be built into the final system (see Appendix A, pp. 138-143), namely, separate language configuration for reading and typing, and means to easily alternate between original and translated text.
4. Prioritized requirements according to the needs of the target users (see Appendix A, pp. 138-143)
5. Avoided potential pitfalls in terms of interface component layout, redesign, and positioning (see chapter 5 – Informing Design, pp. 69-73). The information collected allowed interface components placed where users wanted and expected them on the IM window.

A direct consequence of the benefits listed above was that the UCD methodology employed also reduced the number of evaluation trials needed (see chapter 7 – Evaluation, pp. 101-108), because the system and its interface matched more accurately what users wanted and expected. In most real life projects time and budget are limited resources. Iteration runs and evaluation trials are costly, and time and resource consuming. Hence, reducing the number of iteration runs and evaluation trials is a significant achievement that can deliver higher quality products developed on time and within budget.

6.5. Usability Goals

Completion of the high-fidelity prototype set the stage for testing and evaluation. Following the recommendations by Bevan (1995), the usability goals identified for the evaluation trials were:

- **Effectiveness**: Does the system improve L1-L1 collaboration when participants do not share L1? Can there be collaboration between people that do not speak the same language?
- **Productivity**: Can cross-linguistic IM users complete tasks correctly and completely using their L1?
- **Satisfaction**: Are cross-linguistic IM users satisfied with their interaction with the system? Are the translated messages of acceptable quality, so that users can complete tasks with them?

Usefulness of the system, both perceived and empirically analysed, will be accessed by effectiveness and productivity measurements. The user experience will be a combination of self-reported user satisfaction and a measure of the improvement of the process by which cross-linguistic users perform the same task on standard and cross-lingual IM.

6.6. Summary

The purpose of the high-fidelity prototype was to explore the usability of a cross-lingual IM system developed following UCD principles, with a realistic and familiar system for testing and evaluating in a realistic environment and context of use. Its client-server architecture is appropriate for the distributed and message-based nature of IM. The server, provider of the IM service, mediates conversations and runs the business of managing users, logs, messages, languages and translations. It includes a translation sub-system that is an interface that delegates translation of messages to a dedicated MT system. A client represents an IM user, sending and receiving requests for services to and from the server.

The design of the high-fidelity prototype was determined by the views, preferences and expectations of the target user group. The prototype's GUI was informed by findings from the three information-eliciting methods employed, namely, survey, interview and observation sessions, and log analysis. The interview data provided a comprehensive list of essential functionality that users wanted and expected, together with where in the IM window they expected the interaction components to access this functionality to be. These preferences and expectations determined the design and component layout of the GUI. The survey responses obtained provided many statistically significant results. Of these, preference to type in L1 determined the separate language configuration for reading and typing. The log analysis informed the development of the translation sub-system and improved the quality of the translation outcome. The short length of each message and the large number of emoticons, abbreviations, acronyms, and alternate spellings observed determined a two step translation process: in the first step, acronyms, alternate spellings, and abbreviations from the preferred language of the sender are translated into the preferred language of the recipient; in the second step, the remaining message text is translated by the dedicated MT system.

Throughout the course of the iterative development process, the UCD approach adopted reduced the number of development errors and iteration runs, helped identify and prioritize specific requirements, informed component layout, redesign, and positioning, and reduced the number of evaluation trials needed. Therefore, it optimised budget, time, resource allocation and consumption, yielding a prototype that adequately supported user needs and matched their expectations.

6.7. References

Bevan N. (1995). Measuring usability as quality of use. Journal of Software Quality, 4, pp. 115-130.

Edelstein, Herb. (1994). Unravelling Client/Server Architecture. DBMS 7, 5 (May 1994): 34(7).

Rossum G. V., Lewin L., Willison F. (2001). Programming Python. O'Reilly.

Schussel, George. (1995). Client/Server Past, Present, and Future [on-line]. Available at: <http://www.dciexpo.com/geos/>.

Wiklund, M. (1994). Usability in practice, (editorial), How Companies Develop User-Friendly Products. Academic Press, London.

Word Count: 3691