Translation Server for a Mobile Phone

Clare Scully
B.A. (Mod.) CSLL
Final Year Project, May 2005
Supervisor: Dr. Carl Vogel
Declaration

I hereby declare that this thesis is entirely my own work and that it has not been submitted as an exercise for a degree at any other university.

May 6, 2005

Clare Scully
Acknowledgements

I would like to give special mention to Dr Carl Vogel who has provided constant support and counsel during countless meetings during the entire final year project process. Thank you.

Finally, thanks to my family and friends who have supported me throughout.
“These days, it seems smaller is becoming just as important as faster - at least, when it comes to computing....” - Eric Giguere

“We are in great haste to construct a magnetic telegraph from Maine to Texas; but Maine and Texas, it may be, have nothing important to communicate” - Thoreau
Contents

1 Introduction 9
  1.1 Introduction .................................................. 10
  1.2 Motivation ..................................................... 10
  1.3 Aims ............................................................ 11
  1.4 Overview ....................................................... 11
  1.5 Final Introductory Remarks ................................... 12

2 J2ME Technology 13
  2.1 Introduction .................................................. 14
  2.2 J2ME: Configurations and Profiles ........................... 14
    2.2.1 Connected Limited Device Configuration .................. 14
    2.2.2 Mobile Information Device Profile ........................ 16
  2.3 J2ME Wireless Toolkit ....................................... 16
    2.3.1 Installation ............................................... 17
    2.3.2 Using the Wireless Toolkit ............................... 18
  2.4 Programming with MIDlets .................................... 20
  2.5 Packaging with the Wireless Toolkit ......................... 23
  2.6 MIDlet Deployment .......................................... 25
  2.7 The Translator MIDlet ....................................... 26
  2.8 Sample Emulation ............................................ 26
  2.9 Conclusion .................................................. 32

3 The Database 33
  3.1 Record Management System ................................. 34
  3.2 MySQL ........................................................ 35
    3.2.1 Table Setup .............................................. 36
    3.2.2 Database Population and Tiscali ......................... 37
3.3 Problem with Charsets ........................................ 38
3.4 Problem with Apostrophe .................................... 39
3.5 Conclusion ..................................................... 40

4 Servlets ............................................................. 41
  4.1 Communicating with the Database ......................... 42
    4.1.1 The JDBC Driver ......................................... 42
    4.1.2 The JDBC Optional Package for CDC/Foundation Pro-
        file API .................................................. 43
    4.1.3 Communication via Servlets .......................... 44
  4.2 Servlet Capabilities .......................................... 45
    4.2.1 What is a Servlet? ...................................... 45
    4.2.2 The HTTP Protocol ...................................... 46
  4.3 Hosting Servlets ................................................ 47
    4.3.1 JSWDK .................................................. 47
    4.3.2 Tomcat ................................................ 49
  4.4 Connecting Midlets to MySQL ............................... 51
  4.5 Conclusion ..................................................... 52

5 Populating the Database, a 2nd Approach .................... 53
  5.1 Outline ...................................................... 54
  5.2 Populating the Database via Webpage ..................... 54
  5.3 FORMs ....................................................... 54
  5.4 Processing FORMs ............................................ 55
    5.4.1 PHP .................................................. 55
    5.4.2 Servlets .............................................. 56
  5.5 Conclusion ..................................................... 58

6 Improving Translation Quality .................................. 59
  6.1 Improving Quality ............................................ 60
    6.1.1 Frequency and Success .................................. 60
    6.1.2 Partial Matching ........................................ 61
  6.2 Improving Speed ............................................... 66
    6.2.1 Caching ................................................ 66
    6.2.2 Triggers and Assertions in MySQL .................... 67
    6.2.3 Java: Servlets Integrity Constraints ................ 68
  6.3 Conclusion ..................................................... 69
## Contents

7 Conclusions and Future Work 70

7.1 Summary ................................................. 71

7.2 Achievements ............................................ 72

7.2.1 Problems Encountered .............................. 72

7.3 Future Work .............................................. 73

7.3.1 Incorporating “TextSpeak” ............................ 73

7.3.2 Improving Flexibility ................................. 73

7.3.3 Automata ............................................... 73

7.4 Concluding Remarks ................................. 73

Bibliography ................................................. 75

A Code ......................................................... 77

B Example of Partial Matching .............................. 78

C Tools Utilised ............................................ 81
# List of Figures

2.1 Architecture of J2ME: *Sun Microsystems* ........................................ 15
2.2 The Wireless Toolkit GUI ................................................................. 17
2.3 The Toolkit Emulator ................................................................. 19
2.4 *Project Settings* Dialog Box ...................................................... 23
2.5 MIDlet Packaging, *SUN Microsystems* ........................................... 24
2.6 Sample Emulation of Translator Program, before values are inputted .......... 27
2.7 Step One: Enter phrase to be translated ........................................ 28
2.8 Step Two: Translation is returned ............................................... 29
2.9 Step Three: The user is asked to comment on the success of the translation ................................................ 30
2.10 End screen of Translation Program ............................................. 31
3.1 Initial Translation Table in MySQL ................................................ 37
3.2 Adding a new language to the table ............................................. 38
3.3 Example of data using ‘code’ for special characters ............................. 39
4.1 System Design ............................................................................. 44
5.1 Form Input Webpage ................................................................... 57
6.1 Sample from *most_freq* table. *pos* represents frequency, with entry 1 representing the most frequent translation, 2 the next most frequent, etc. *id* identifies the translation row in question .......................... 67
B.1 Partial Matching 1 ....................................................................... 79
B.2 Partial Matching 2 ....................................................................... 80
Abstract

This project describes the steps and tools described in setting up a translation program for a mobile phone. The aim is to demonstrate the concept of creating a service where a server receives a SMS message in one language and returns a translation of this message to the user in a different language they request. The program is written in J2ME, MySQL is used to create the database and servlets are used to link the two together.

While this project is an extension of someone else’s final year work, I also hope that it can be used by students as a foundation for future work in this area. It is documented accordingly.
Chapter 1

Introduction
1.1 Introduction

This project implements a translation server for a mobile phone. This report details the steps involved in setting up such a server and the tools necessary for such a task. In America today there are over 3000 translation companies in existence, as well as numerous individual translators (Schulman, 2004). The demand for translation services is huge. The proliferation of online translation services is well documented, but as of yet no one has properly constructed a system for accessing a translation service from a mobile phone.

1.2 Motivation

A survey published recently (Vodafone.ie) on active mobile phone users shows that roughly three and a half million people in Ireland use mobile phones on a daily basis. The huge growth of the mobile phone industry over the last decade serves to underline the unrelenting demand on the mobile phone market, and the constant need for newer, more up-to-date mobiles.

Another survey, carried out in December 2004 by Púca, one of the main providers of mobile messaging in Europe, revealed that there is a also a very strong demand in Ireland for services accessible via mobile phones (Puca, 2004). While the survey is concerned more with government services such as offering reminders about CAO results and NCT tests, it nevertheless underlines the amount of people that are willing to use services offered to them on their mobile phones, which is fully understandable in a society in which is almost dependent on these handheld devices. The survey explains that for many people sending a text message is much more convient that using a computer. Thus, a clear market for applications accessibly via a mobile phone that were previously only available on a computer.

The phrase book has always been a near essential to tourists travelling abroad, millions of them are sold every year. Although phrase books and translator applications are available online and can be purchased as special software, to date not many phrase book applications are available for a mobile phone.
CHAPTER 1. INTRODUCTION

1.3 Aims

This program aims to replicate the idea of a phrase book for a mobile phone. As it builds on a project from a previous year, a feasible design framework for the translation server was already proposed. I hoped to implement and add to this framework. I aimed to provide a fairly comprehensive English/French, French/English translation server with the hope that once this was in place replicating the system for other languages would be relatively straightforward. Thus, design generalisation and redeployability was an aesthetic.

1.4 Overview

This dissertation is presented in five main parts:

- **Chapter 2:** Discusses J2ME (Java micro-edition), the configurations and profiles associated with it and the tools necessary to build applications in J2ME (notably SunMicrosystems J2ME Wireless Toolkit). It describes the functionality of the Wireless Toolkit, the code used in the translator application and provides a graphical description of sample emulation of a translation.

- **Chapter 3:** Discusses the use of MySQL in the project, the way in which the database was set up to facilitate the translation process and some of the problems encountered when setting up the translation table.

- **Chapter 4:** Discusses the way in which the mobile phone will connect to the database to retrieve the data necessary for carrying out translations. This includes a description of JDBC drivers and servlet technology.

- **Chapter 5:** Discusses a system for updating and altering the database via the Internet that will allow clients to add data to the database, as well as the database administrators.

- **Chapter 6:** Discuss the frequency and success ratings associated with a translation and the implementations of a partial matching system and a caching system in the project.
• **Chapter 7:** Summarises the information given in the preceding chapters and directives for future work are given in the hope that this project can be used as building block for future work.

### 1.5 Final Introductory Remarks

This dissertation represents a synthesis of my undergraduate experience, and while flaws are noted where appropriate, so are novel contributions. I highlight the areas in which I motivated my learning in the project by areas outside my degree syllabus. I also highlight how the nature of the degree program informed its progress.
Chapter 2

J2ME Technology
2.1 Introduction

In today’s growing mobile phone market more and more phone owners are looking to develop their own software (games, etc.) for their mobiles, and anecdotal evidence suggests that parents increasingly want their children to have mobile phones so that the whereabouts and well being of their children can be established. With Java 2 Micro Edition (J2ME), the Connected Limited Device Configuration (CLDC) and the Mobile Information Device Profile (MIDP) programmers can write applications for use on hand-held devices such as a PDA or a mobile phone. Because J2ME is a limited version of java developed for writing applications for handheld devices it is both simple and versatile. It contains three basic layers — the configuration layer, the profile layer and the virtual machine. While J2ME programming is very similar to writing applets in Standard Edition java the steps involved in deploying a J2ME application can be different to those involved in compiling and running standard java code. The following sections provide a brief description of CLDC and MIDP (the configuration and profile used in creating applications for use on a mobile phone), and the steps involved in installing the software necessary for writing and compiling code in J2ME.

2.2 J2ME: Configurations and Profiles

2.2.1 Connected Limited Device Configuration

The Connected Limited Device Configuration (CLDC) is a limited version of the Connected Device Configuration (CDC). As J2ME applications are used on such a wide variety of devices, configurations are used to handle the different capabilities of these devices. Thus, a J2ME configuration defines the virtual machine and low-level API that is made available on a particular device, providing the development foundation.

The CDC is used on devices that have powerful processors and fairly large amounts of memory available. The CLDC, on the other hand, is more suitable for use with a device such as a mobile phone, as the mobile would be unable to meet the requirements of the CDC. The CLDC uses a virtual machine that is designed to run in an environment where limited amounts
of memory are available to it, known as the KVM or the K Virtual Machine (because it’s size is measured in kilobytes). The KVM is similar to the JVM (Java Virtual Machine) except that it lacks some of the features. For example, you can’t use floating-point math with the KVM, and it is necessary to pre-verify source code before compilation. The KVM provides a limited core Java language set and is written in the C programming language.

The CLDC is used on devices with the following:

- Minimum of 160 KB total memory, 128 KB of which is needed for KVM and the runtime libraries
- Minimum 16-bit 25-MHz processor
- Limited user-interface facilities
- No (or intermittent) low-bandwidth network connection
2.2.2 Mobile Information Device Profile

The Mobile Information Device Profile provides a framework within which to work. Basically, it defines the unique requirements for each device within a certain configuration. In this way it defines the set of API’s that are available to the J2ME application.

There are two main profiles that have been developed by SUN for use on all platforms:

- The Foundation Profile
- MIDP

The Foundation Profile is used, for the most part, with the CDC configuration and MIDP is used with CLDC (It is specifically designed for writing applications for mobile phones and pagers). MIDP supports a limited selection of J2ME’s class library, including classes for input and output.

The MIDP package requires:

- an Input/Output device such as a telephone pad or touch screen
- 128Kb of non-volatile memory to run components
- 8Kb of non-volatile memory for gathering and storing data
- 32Kb of volatile memory to run java

2.3 J2ME Wireless Toolkit

The J2ME Wireless Toolkit is a toolkit developed by SUN Microsystems for creating applications for use on mobile devices. It provides build tools, utilities and an emulator on which to run the applications developed. These applications must be compliant with the ‘Java Technology for the Wireless Industry’ (JTWI, JSR 185) specification. Programs for CLDC and MIDP can be developed with the Wireless Toolkit. The version used in this project is 2.2, a release that has recently\(^1\) won the ‘Jolt Product Excellence Award’

\(^1\)January 2005.
and the ‘Developper.com Product of the Year Award’. The Wireless Toolkit 2.2 provides:

- Support for new API’s
- Support for Multimedia Messaging Service (MMS)
- A simulated Bluetooth environment
- Network support for MMS and Bluetooth (This allows Java-enables devices to integrate with a Bluetooth environment)

The Wireless Toolkit provides a GUI environment interface to avoid all command line commands that would be necessary if the CLDC and MIDP were downloaded separately. The toolkit allows the CLDC and MIDP to work together forming a java runtime environment (JRE).

### 2.3.1 Installation

A version of the Java Development Kit (JDK) must be obtained before the Wireless Toolkit can be installed. The JDK enables compilation of java code. jdk1.5.0 can be downloaded at java.sun.com\(^2\) (but any JDK will do as long as it is version 1.3 or newer). Once installed, the computer must be told where it can find Java. This is acheived by setting the environmental variables.

---

\(^2\)Last verified: April 2005.
Setting Environmental Variables

In Windows XP this is achieved by clicking on System in the Control Panel. Click on the Advanced tab. There is a button for Environmental Variables at the bottom. Click on this. A window split into two sections, User variables and System variables will appear. In the User variables add a new variable. Call this JAVA_HOME and in the Value section put in the address of the jdk

(e.g. C:\Program Files\Java\jdk1.5.0)

In the System variables section click on the variable Path and click Edit to change the value. A new window will pop up called "Edit System Variable" where "Variable name:"=Path. In the "Variable value" field enter a semi-colon after the existing values and then type in the path to the java bin directory

(e.g. C:\Program Files\Java\jdk1.5.0\bin)

Then click on Ok, exit the System Properties window and restart the machine.

The same can be achieved on Linux computers operating in the Mandrake environment by adding the required classpath to the .bashrc file.

(e.g. export CLASSPATH=$CLASSPATH:/usr/local/java:. )

Once this has been done, the Java Wireless Toolkit can be downloaded from

http://java.sun.com/products/j2mewtoolkit/download-2_2.html

Obviously, these exact directories may change over time.

2.3.2 Using the Wireless Toolkit

There are five main steps involved in developing applications with the Wireless Toolkit. These are:

1. Opening the KToolbar
2. Creating a project - this is achieved by clicking on the Create Project button. There are no restrictions on the name given to the project but the name for the MIDlet suite must be the name of the main MIDlet class used in the project (the one that runs the program).

3. Verifying that the project properties are correct

4. Writing the code for the MIDlet and saving it in the src folder

5. Clicking on build to compile the code and on run to run the program in the emulator

When a project is created in the toolkit, a number of directories are automatically created. These are:

- bin (contains JAR - Java Archive - and JAD files)
• classes (contains any third party libraries necessary)
• res (contains resources used in the application such as images, etc)
• src (contains the source code)
• tmpclasses (holds temporary files)

2.4 Programming with MIDlets

What is a MIDlet?

A MIDlet is basically the J2ME version of an Standard Edition Java applet. When programming in MIDlets the base MIDP application class is extended and the two main packages:

• javax.microedition.midlet (this includes the application’s base MIDlet class)
• javax.microedition.lcdui (provides GUI elements for use in the application)

are imported. The MIDlet class provides three methods, startApp(), pauseApp(), and destroyApp() that are used to initialize, pause and end the application, respectively. The startApp() method is called after the constructor and anytime the application is made active. The destroyApp() method is called by the device’s application manager when an application is about to be shutdown. Most “cleanup” code is placed here. pauseApp() is a method developed to deal with situations where the application is no longer active, perhaps because the user has switched to a different application and the MIDlet is no longer able to run.

Most of the User Interface elements at the developers disposable with the lcdui package are identical to SWING components and include elements such as TextBoxes, Alerts, ImageItems, etc. The commandListener servers as a listener for any commands the application might contain (e.g. Exit, Translate, etc.).

All these UI items are managed by a Display object. This object enables setting the focus to a particular element in the application, and obtaining information about the capabilities of the device in question.
MIDlet Files - JAR and JAD

JAR
The Java Archive file format (JAR) is used to bundle together multiple files such as class files and resources associated with applications into a single archive file. It is based on the ZIP format. In terms of MIDlet applications the JAR file is the packaged version of the application that will be downloaded onto a mobile phone for use by a user. In this way, the application and all the images and resources associated with it can be downloaded to the phone in a single transaction. Therefore, the user does not need to reconnect multiple times. JAR is written in java and is the only archive format that allows the bundling together of audio files, image files and class files together in a single archive. The JAR file format has many benefits associated with it. It provides:

- Security
- Decreased download time
- Compression
- Packaging for extensions
- Package sealing
- Package versioning
- Portability

Every JAR has a manifest file associated with it. This provides information about the classes contained in the JAR file. The following information must be in the JAR file for the application manager to accept it:

- MIDlet-Name
- MIDlet-Version
- MIDlet-Vendor
- MIDlet-number
JAD

A Java Application Descriptor File (JAD) contains additional configuration information about the application. The JAD can be accessed at runtime using the `MIDlet.getAppProperty` method. Basically, the JAD provides information about the contents of a JAR file for the application manager. From this the application manager can determine if a MIDlet is suitable for running on a particular device. The JAD also allows for parameters to be passed to the MIDlet without changes to the JAR file being necessary. It contains a predefined set of attributes that allow the application manager to setup, run and install MIDlets.

Like the JAR, the JAD has six required attributes:

- MIDlet-Name
- MIDlet-Version
- MIDlet-Vendor (can be ‘Unknown’)
- MIDlet-Jar-URL
- MIDlet-Jar-Size (in bytes)

The JAD file is not required by the specification, but most handsets won’t accept a JAR, instead they require the JAD and install what’s specified in the MIDlet-Jar-URL field.

Some of the above attributes described above are required by both the JAR and the JAD. The values for these attributes must be identical in each.

You can define the above and thus the contents of the manifest file by using the `Project Settings` dialog box in the Wireless Toolkit.
2.5 Packaging with the Wireless Toolkit

Another advantage to creating applications using the Wireless Toolkit is the ease with which developers using the program can package their applications. Packaging is important because it is required if the application is to be run
from a web service or, as in this case, on a hand-held device. There are two choices when creating a package using the Wireless Toolkit:

- **Create Package**: MIDlet suite descriptor and JAR are generated and placed in the bin directory of your project.

- **Create Obfuscated Package**: This shrinks the size of the JAR file. Obfuscation removes class information such as local variable names that are not really important. Using this option, classes and methods are renamed, making them ambiguous. This means that the outputed package cannot be decompiled, and thus obfuscation protects the source code. To use this feature a code obfuscator plug-in must first be obtained.

The recommended code obfuscator plug-in is the retroguard.jar which can be downloaded at:\n\nhttp://www.retrologic.com

Once downloaded, the file must be placed in the bin directory of the Wireless Toolkit to use it.

After packaging, the MIDlet suite can also be signed. This is an optional

---

\(^4\text{Last verified: April 2005.}\)
step and is in no way required. Signing the suite creates a digital signature for the JAR file, which is then added to the JAD file. To sign the suite using the Wireless Toolkit simply click on the sign option in the Project menu.

2.6 MIDlet Deployment

There are a few ways of deploying the MIDlet suite package onto a mobile device once the application is stable and has been tested on the Wireless Toolkit emulator:

- Via the Web: The package can be uploaded onto a web server and downloaded to the mobile phone via wap
- Via USB/serial/irDA connection: This option often requires special software
- Via Infared or Bluetooth

However before actually downloading and running the application onto a mobile device, the application should be tested using OTA provisioning. Over the Air Provisioning Mode is used when a MIDlet suite has been created from packages MIDlets, JAR and JAD files and tests the functionality of the application. The Wireless Toolkit provides simulated OTA provisioning. This means that the suite is packaged into JAR and JAD file format, deployed to a provisioning server and downloaded to an emulated device. Running an application through OTA provisioning differs in many ways from simply testing the application on the emulator. OTA ensures:

- Package validation
- Authentication of the MIDlet suite
- That permissions to access sensitive APIs are set
- Push functionality

OTA provisioning mode can be run by choosing the Run via OTA option in the Project menu. Running the application in this way simulates deployment from a web server. A default emulator device with a graphical Application
Manager System (AMS) is used, that is ready to install the application. (An AMS is the software on the device that manages the downloading and lifecycle of a MIDlet). When the Emulator appears, click on the Apps button and choose Launch from the menu. The URL of the application will be displayed. Then select Go from the menu. The .jad file of the application will be presented. The user is given the option of downloading it. Then details about the application will be presented. Select Install. If there are no problems installing the application it can be run on the emulated device to verify that it functions correctly.

2.7 The Translator MIDlet

The Translator MIDlet is the code behind the program that appears on the mobile phone. In the application, the user is given instructions and told to click on ’Next’ to continue. When ’Next’ is clicked, a new screen appears with three text boxes on it. The user is asked to enter the sentence to be translated in the first of these boxes, the source language in the second and the target language in the third. The user then clicks on the Translate option to obtain a translation of the sentence. This translation is returned on a new screen and the user is asked whether the translation was useful or not (see Section 6.1.1 for more details). They respond by selecting yes/no from a menu. The Translator MIDlet controls all of this.

For a more detailed description of the code used in the MIDlet see Appendix A.

2.8 Sample Emulation

The screenshots displayed on the following pages show an emulation of the translation of the phrase “Where are the toilets” from English to French.
Figure 2.6: Sample Emulation of Translator Program, before values are inputted
Figure 2.7: Step One: Enter phrase to be translated
Figure 2.8: Step Two: Translation is returned
Figure 2.9: Step Three: The user is asked to comment on the success of the translation
Figure 2.10: End screen of Translation Program
2.9 Conclusion

In this chapter, the specifications of CLDC and MIDP were briefly discussed and it was explained how they are necessary in the development of applications for use on a mobile device. The J2ME Wireless Toolkit interface was described with special mention given to packaging of the application and the contents of the resulting *.jar and *.jad files. The functionality of the J2ME interface was also described, both in text and graphical form.
Chapter 3

The Database
The translation system has been implemented using a database that stores complete sentences and their equivalents in various different languages. As the aim is to design a system that implements a computational version of a traditional phrase book, it is necessary that whole sentences and their translations are stored somewhere that is accessible via the mobile phone. Using a database allows easy storage and retrieval of this data.

Databases are an invaluable tool for storing information. Computerised databases facilitate the linking of related tables/lists of data, searching, sorting and querying information. The following sections detail two different database implementations:

1. Record Management System
2. MySQL

## 3.1 Record Management System

In previous work on this topic (Madill, 2004), it was suggested that one way of storing the possible translatable sentences would be to store them on the actual device (the mobile phone). This would be achieved using the Record Management System (RMS), an API that is basically a subsystem of MIDP that allows on-device data persistence.

While this sounds fairly reasonable there are two main problems with such an approach. The first is that of available memory. The amount of memory available for record-based data storage varies from device to device, although there is a minimum of at least 8K of non-volatile memory available, and RMS provides methods for checking how much memory for data storage remains. Although this project hopes merely to provide translations for phrases useful for the average tourist, and thus is somewhat limited in the amount of sentences that need to be stored, were the project to be expanded at a later stage, the infinite combinatorial possibilities of language must be taken into account. Any comprehensive translation database must, by nature, be huge, and the RMS system simply does not meet such a requirement. Nor does it allow for alteration to the records, excepting an unlikely situation where the user is willing to re-download the program to their mobile phone every time
the database is updated or altered in some way. This is the second problem with RMS. Translation systems can always be improved upon, therefore a more flexible storage system is required.

RMS does, however, have one main advantage over other possible implementations, the translations are stored on the phone itself, thus it is not necessary for the user to connect to an outside server to use the program. As such a connection (via WAP) is a paying service, perhaps there is some merit to such an implementation. A smaller program, that perhaps only translates phrases used in emergency situations between a limited number of languages, implemented using RMS is easy to envisage.

More information on RMS can be found at: http://developers.sun.com/techtopics/mobility/midp/articles/databaserms

The other alternative to RMS is to use an external database. Section 2.3 will look at one such external database server MySQL.

### 3.2 MySQL

MySQL is one of the most widely used database servers in the world today, with many high-profile users include NASA and Yahoo!. It is open-source and available for use to anyone. It runs on many different platforms (Windows, Linux, Mac OS X, Unix, etc). The version of MySQL used in this implementation (4.0.23) came as part of the EasyPHP1-7 package (see Appendix C for installation details). EasyPHP is a program that incorporates Apache, PHP and MySQL in one package. (MySQL is also available on the TCD macneill server.)

MySQL is an implementation of the relational database model where relations are organised into tables. This suits the translation server purpose, because of the relation that exists between phrases and their foreign language equivalents.

\[1\text{Last verified: April 2005.}\]
Every MySQL server consists of a number of databases, each database consisting of a number of tables. A table consists of rows and columns. It can be empty but must have at least one field associated with it, i.e. at least one column must be created. By executing a CREATE command a table is created and the database server is told how many columns it will have, the names of these columns and the type of values they will contain (INTEGER, VARCHAR, etc). Using INSERT, ALTER and UPDATE commands you the table can be populated. (In EASYPHP this is unnecessary. As there is a HTML interface provided, it is not necessary to execute the SQL commands via the Command Prompt.) SELECT statements can then be used to query the database and retrieve necessary information.

### 3.2.1 Table Setup

As was said above, a table has a number of columns and rows. A simple database containing only one table was used. The columns used in the initial table (Figure 3.1) were `id`, `english` and `french`. (The table later become more complex as new columns to facilitate new languages were added in). Thus each translation is associated with an `id` number. This is necessary because of the notion of PRIMARY KEYS used in MySQL. A primary key ensure that at least one column uniquely identifies each record in the table. A primary key must be unique and cannot be null. Although in this particular case `english` could be used as the primary key, this would become problematic if there were two French translations for the same English sentence. It would become even more of a problem later on, as new languages are added to the database, because if `english` was the primary key, then some value would always have to be inputted to the field. Thus even if you were adding in an Italian translation of a French sentence, you would also have to input something for the English value. Hence we use `id` as the primary key.

The `id` field contains values of type INTEGER. The `english` and `french` columns hold values of type VARCHAR. VARCHAR is the MySQL equivalent of a String. The length of each column has been specified to 160 characters (VARCHAR(160)) because the standard no of characters in an SMS message is 160.

MySQL is very flexible and it is easy to add new columns to the table by
executing an ALTER table command. Thus the following:

```
ALTER TABLE tr1 ADD german VARCHAR(160);
```

adds a new column, ‘german’ to the database (see Figure 3.2).

### 3.2.2 Database Population and Tiscali

As the project aims to represent a phrase book, data collected from an online phrase book found at[^2] [http://www.tiscali.co.uk/reference/phrases/](http://www.tiscali.co.uk/reference/phrases/) was used to populate the database. Tiscali offers a number of phrases grouped into categories such as Accomodation, Eating Out, Emergencies, Doctor, Pharmacy, etc. in French, Spanish, Italian and German. It is mostly data from the French Phrase Finder that was used, as the goal was to provide a fairly complete database in one language before adding in others. A small selection of phrases from Tiscali’s Spanish and Italian sections are also used.


<table>
<thead>
<tr>
<th>id</th>
<th>english</th>
<th>french</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Is there a hotel here?</td>
<td>Il y a un hôtel ici?</td>
</tr>
<tr>
<td>2</td>
<td>Do you have any vacancies?</td>
<td>Vous avez des chambres?</td>
</tr>
<tr>
<td>3</td>
<td>I’d like a room</td>
<td>Je voudrais une chambre</td>
</tr>
<tr>
<td>4</td>
<td>How much is it?</td>
<td>C’est combien?</td>
</tr>
<tr>
<td>5</td>
<td>Have you anything cheaper?</td>
<td>Avez-vous quelque chose de moins cher?</td>
</tr>
<tr>
<td>6</td>
<td>Are there any other hotels nearby?</td>
<td>Il y a d’autres hôtels dans le coin?</td>
</tr>
<tr>
<td>7</td>
<td>Where can I change some money?</td>
<td>Où est-ce que je peux changer de l'argent?</td>
</tr>
<tr>
<td>8</td>
<td>How do I get into town?</td>
<td>Pour aller en ville?</td>
</tr>
</tbody>
</table>

Figure 3.1: Initial Translation Table in MySQL
Figure 3.2: Adding a new language to the table

Rather than sitting at a computer and adding in each phrase to the database manually, a java servlet was written to read the phrases from a text file, create INSERT statements from the phrases and then insert them into the database.

3.3 Problem with Charsets

One major problem with this version 4.0.23 of MySQL (and the version provided on macneill) is that you cannot specify the charset to be used. This means that when one tries to retrieve entries from the database difficulties will be encountered when retrieving any of the characters used in foreign languages (e.g. à, à, é, etc.) They will simply not be returned to the user.

A type of ‘code’ has been used to counteract this problem. Every special character (used in European languages) has been given an equivalent code, and it is the value of this code that is entered into the database instead of the special characters. The code is then converted back into à, à, é, etc. by the java servlet that communicates between the mobile phone and the database.
Also, because a translation is not necessarily carried out on an English sentence, when carrying out a translation, any special characters in the phrase to be translated must be converted into the code before any queries to the database are carried out.

<table>
<thead>
<tr>
<th>Character</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>à</td>
<td>a1</td>
</tr>
<tr>
<td>á</td>
<td>a2</td>
</tr>
<tr>
<td>á</td>
<td>a3</td>
</tr>
<tr>
<td>è</td>
<td>e1</td>
</tr>
<tr>
<td>è</td>
<td>e2</td>
</tr>
<tr>
<td>á</td>
<td>e3</td>
</tr>
</tbody>
</table>

Table 3.1: Sample of code for special characters

(For entire listing of special characters and their equivalent codes consult Appendix C).

<table>
<thead>
<tr>
<th>id</th>
<th>english</th>
<th>french</th>
<th>german</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Is there a hotel here?</td>
<td>Il y a un hôtel ici?</td>
<td>NULL</td>
</tr>
<tr>
<td>2</td>
<td>Where can I change some money?</td>
<td>Où 1 est-ce que je peux changer de l’argent?</td>
<td>NULL</td>
</tr>
</tbody>
</table>

Figure 3.3: Example of data using ‘code’ for special characters

3.4 Problem with Apostrophe

Similarly there is a problem when inputting words containing apostrophes into the database that must be resolved. Because SQL identifies a VARCHAR variable as the string delimited by a single quotation mark (‘) on
either side, entries such as ‘I’m going to the restaurant’, etc. are troublesome. The SQL will parse ‘I’ as a VARCHAR, but as it is unable to recognise the rest of the sentence, ‘m going to the restaurant’, it will return an error.

Therefore, it is necessary to place a backslash (\) before the apostrophe when entering sentences such as the above into the database:

\texttt{INSERT INTO table(english) VALUES('I\'m going to the restaurant');}

However, it is not \texttt{I\'m} that is entered in the database but simply \texttt{I'm}, the backslash is used as a mechanism to allow SQL to recognise special characters.

Similarly, we must also deal with apostrophes when carrying out a translation. The sentence to be translated may contain an apostrophe that will have to be converted before querying the database (as with the special characters in Section 3.3). This is carried out by searching the phrase to be translated, identifying the string “'” and replacing it with “\"”. Despite using two backslashes here, only one will remain in the resulting string.

### 3.5 Conclusion

In this chapter the motivations for using a database in the project have been discussed. Record Management System (RMS) and MySQL have been proposed as possible implementations, the advantages and disadvantages of both approaches have been explained, along with the reasons as to why MySQL has ultimately been the software used. The setup of the translation table in the database has been described, both in text and graphical form and some of the problems encountered, especially those involving charsets, have been discussed.
Chapter 4

Servlets
4.1 Communicating with the Database

The following chapter discusses the steps involved in getting the MIDlet code to interact with the MySQL database server.

4.1.1 The JDBC Driver

As stated previously direct communication between java and a database is not possible. To enable communication between the two a JDBC driver must be used. A JDBC driver is a Data Access API. Many JDBC drivers are available and tailored to suit the needs of different types of databases - e.g. MySQL, Oracle, MS Access, etc.). The JDBC driver most commonly used for accessing MySQL databases is the MySQL Connector/J and is available from the MySQL website at:\footnote{Last verified: April 2005.}


The MySQL Connector/J is the official JDBC driver for MySQL. It is an implementation of Sun’s JDBC 3.0 API for MySQL. The driver can be used in any java environment supporting the JDK1.2 or higher, and it can be used with any version of MySQL.

Once the MySQL Connector/J has been downloaded to install it the classpaths must once again be set. This can be done in three different ways:

1. Unzip the archive and copy the \texttt{com} and \texttt{org} directories to somewhere in the computer (it doesn’t matter where). Then put the directory holding \texttt{com} and \texttt{org} into the classpath.

2. Alternatively the archived file (mysql-connector-java-[version]-bin.jar) can be placed into your classpath

3. The archive file can also be placed into the \$JAVA\_HOME/jre/lib/ext directory. By doing so the classpath to the driver will automatically be set.

If choosing to unzip the archive, note that it is the \texttt{com} directory that will be used for accessing the driver as this directory contains the \texttt{com.mysql.jdbc.Driver} class that implements \texttt{java.sql.Driver}.\footnote{Last verified: April 2005.}
Using the `java.sql` class

When writing a java class to interact with the MySQL database import the `java.sql.*` package at the beginning of the file. To create a connection to the database it is necessary to create a number of objects. These are:

- **Connection object**: Creates a connection with the database
- **Statement object**: SQL queries to be executed are done so using the Statement object
- **ResultSet object**: This holds the results of an SQL query to the database
- **ResultSetMetaData object**: Used to get information about the types and properties of the columns in a ResultSet object

To initialize an instance of the driver the following code is used:

```java
Class.forName("com.mysql.jdbc.Driver");
```

and to get a connection to the driver the following syntax is used:

```java
Connection conn=DriverManager.getConnection("jdbc:mysql:<address-of-database><userName&password-if-required>");
```

e.g. `Connection conn=DriverManager.getConnection("jdbc:mysql://localhost/scullyc_db?user=scullyc&password=enterDB");`

### 4.1.2 The JDBC Optional Package for CDC/Foundation Profile API

The JDBC Driver allows communication between java classes and database servers. It does not however facilitate communication between the database server and MIDlet files. There is a package provided by SUN Microsystems called the JDBC Optional Package that does enable such an operation. The package provides a subset of the `java.sql` package that is appropriate for such a communication. However, the package is only for use with applications in J2ME that are written using CDC and the Foundation Profile API. Because a mobile phone does not support CDC, the optional package cannot be used in this project. As of yet, no JDBC package for CLDC exists.
4.1.3 Communication via Servlets

Taking this into account, the only way to facilitate communication between MIDlets and an MySQL database is to use a servlet that can access both MIDlet files and the JDBC driver and transfer data between the database server and the J2ME code. Servlets will be discussed in detail in the next section.

Figure 4.1: System Design
4.2 Servlet Capabilities

4.2.1 What is a Servlet?

A java servlet is a program that runs off a web server. Servlets are similar to applets except that they are run in a server application to answer client requests. Servlets are mostly used for processing and storing data submitted by a HTML form, for providing dynamic content such as the results of a query to a database and for managing state information on top of the stateless HTTP.

A servlet is an instance of a class that usually extends one of the two standard implementations of the javax.servlet.Servlet interface. These are:

- javax.servlet.GenericServlet
- javax.servlet.HttpServlet (a subclass of javax.servlet.GenericServlet)

A servlet is initialised when the Servlet class is loaded by a server application, creating an instance of the servlet by calling the no-argument constructor. The servlet’s init(ServletConfig config) method is then called. When the servlet is initialized, the service(ServletRequest req, ServletResponse res) method is called when a request is made to the servlet.

Servlets are not the only server-side applications that can be used to handle server requests/responses (others include PHP, ASP, ColdFusion, etc.) but there are numerous advantages to using servlets:

- Persistence: Once a servlet has been loaded into the server’s memory it stays there as a fixed Java object
- Efficiency: When servlets carry out request operations there are no variables to instantiate (excepting the first time the servlet is run). Thus they are very efficient
- Better server interaction: Because servlets are tightly integrated with the server, server interaction is better and more sophisticated than is possible with other implementations such as CGI.
4.2.2 The HTTP Protocol

The HyperText Transfer Protocol (HTTP) is the protocol used by a browser to send a request to a Web Server. HTTP is a request-response oriented protocol. A request in the HTTP Protocol is made up of a request method, URL, a header and the body of the request. A response contains a header, the body and the result code.

The `service(ServletRequest req, ServletResponse res)` method of HttpServlet recognises six different HTTP request methods, but only two of these are relevant for our purposes. They are:

- GET
- POST

Both GET and POST send information to the servlet but do so in slightly different ways.

**GET vs POST**

GET sends the information to the servlet encoded in the URL, whereas with the POST method data is sent to the servlet as standard input.

Because GET sends the data in the URL, it is not very secure and the value entered is visible on the screen. GET queries are most commonly used by search engines. This type of query is useful in such a situation because it can be bookmarked, copied and pasted into e-mail, etc.

The POST method is more secure than the GET method as the data is not transmitted via the URL, but instead as a separate stream. It also has the added advantage that query length is unlimited. With the GET method query length is limited to a certain number of characters. This can become troublesome because the server may truncate data that exceeds this number of characters.

Once the request protocol (either GET or POST) has been chosen, the servlet obtains the information to be processed by carrying out either the
doGet(HttpServletRequest req, HttpServletResponse res) method or the do-
Post(HttpServletRequest req, HttpServletResponse res) method.

4.3 Hosting Servlets

To test a servlet it must be uploaded it onto a web server. But before this
is done the code must be compiled. To do this the Servlet API classes are
necessary. The Java Servlet Specification 2.1 is available at


This contains the classes and source files for the javax.servlet and javax.servlet.http
packages.

To use the packages add the servlet-api.jar file to the classpath. This is down
in the same way that the jdk1.5.0 was added to the classpath in Section 2.3.1.
In the Environmental Variables dialog box, click on Path, add a semi-colon
after the last value and then add in the address of the package.

Alternatively, the code can be compiled from the same directory where the
servlet-api.jar package is stored.

However, it is not necessary to download the Servlet API classes on their
own, because they are included in various development kits available for the
compiling and testing of java servlets. The two most common development
kits, the JavaServer Web Development Kit and Apache Tomcat, are both
suitable for the purposes of this project.

4.3.1 JSWDK

The JavaServer Web Development Kit is a kit provided by SUN for develop-
ing and hosting servlets. It contains the Servlet API classes and tools and
can be downloaded at


---

2Last verified: April 2005.
3Last verified: April 2005.
Using JSWDK

Create a new folder in the base jswdk directory for storing the servlet. For example, create a Translator directory:

C:\jswdk-1.0.1\Translator

Create a WEB-INF directory inside this folder. Add a servlets folder and copy over the mappings.properties, servlets.properties, webapp.properties and mime.properties files from the

C:\jswdk-1.0.1\examples\WEB-INF

folder. Store the compiled servlet code in the servlets directory. The servlets.properties file must then be edited. For example, if a servlet called Translator.java was added into the servlets directory, the servlets.properties file must be edited to include the line:

Translator1.code=Translator

This is because all entries into this file must be of the format:

<servletname>.code=<servletclass>

In the above example, Translator1 is the the name that will be used in the URL address of the servlet and Translator is the name of the compiled java class. If the servlet requires parameters, this information must also be added to the servlets.properties file, in the format:

<servletname>.initparams=<name=value>,<name=value>,etc.

This new application can be added to the server by editing the webserver.xml file in the JSWDK root folder. To use the Translator example above the following would be addd in:

<WebServer id="webServer">
  <Service id="service0" port="2207">
    <WebApplication id="Translator" mapping="/Translator"
      docBase="Translator"/>
  </Service>
</WebServer>
Here I have specified that port 2207 is to be used when running the server. This is not necessary, and the default port for use is 8080.

To run the server click on the `startserver` file in the root directory. The `Translator` servlet can now be accessed via the address:

- http://localhost:2207/Translator/servlet/Translator1,

or alternatively

- http://127.0.0.1:2207/Translator/servlet/Translator1

Note that the address if `/Translator/servlet/Translator1` even though the servlet class is saved in a `servlets` folder.

When testing servlets using the JSWDK (or indeed any server), remember that when changes have been made to the code and this code has been re-compiled, it is necessary to shutdown and restart the server before the server can recognise that changes have taken place.

### 4.3.2 Tomcat

Tomcat contains code that is based on JSWDK. It is part of the Apache Web Server family and is opensource. It can be downloaded from:

http://jakarta.apache.org/tomcat/index.html

Like JSWDK, Tomcat is a Web Application container, but Tomcat is more widely used as it is the official Reference Implementation for the Java Servlet and JavaServer Pages(JSP) technologies.

Tomcat will work with any Java Development Kit of version later or equal to JDK 1.2. Once tomcat has been downloaded, go to the `jakarta-tomcat-version` folder and then run the file `setclasspath.bat`. This should automatically set up your system variables (providing classpaths to the JDK have already been set).

---

4Last verified: April 2005.
To start Tomcat, click on the `startup.bat` file. To ensure that it is running properly access:

`http://localhost:8080/`

If the Tomcat homepage is displayed, then Tomcat is running correctly.

**Using Tomcat**

Servers that conform to the Servlet API Specification 2.2 are required to accept Web Application Archive (WAR) format. To facilitate this, the directories of an application developed under Tomcat must be laid out in a certain fashion. The document root directory of a web application is the top-level directory of the program. This directory should be created inside the `webapps` folder found in the root folder of Tomcat and the files found inside the document root directory should be organized in the following way:

- `/WEB-INF/web.xml`: This is where the servlet is described, along with any initialization parameters and container-managed security constraints necessary.

- `/WEB-INF/classes`: Where the java classes are stored.

- `/WEB-INF/lib`: This directory stores all JAR files that are used in the servlet. These JAR files contain java classes and examples include a JDBC driver.

- `.jsp, .html, etc`: All html and jsp classes are also stored in the document root directory.

The `web.xml` file mentioned above is the Web Application Deployment Descriptor and as such is very important. This descriptor is an XML file and gives the server all the information about the application that is necessary for its deployment. When creating a new application, the easiest way to create a Web Application Deployment Descriptor is simply to copy over the `web.xml` file in the `servlet-examples` directory and edit it appropriately. Servlets to be deployed in the application are described in the `web.xml` file as shown in the following example (where `HelloWorldExample` is to be replaced by the name of the servlet used in the application):
<servlet>
   <servlet-name>HelloWorldExample</servlet-name>
   <servlet-class>HelloWorldExample</servlet-class>
</servlet>

It is also necessary to map the servlet to a certain URL. This is done by adding a <servlet-mapping> tag to the web.xml file, as is shown below:

<servlet-mapping>
   <servlet-name>HelloWorldExample</servlet-name>
   <url-pattern>/servlet/HelloWorldExample</url-pattern>
</servlet-mapping>

This means that the HelloWorldExample servlet can be accessed by the address:

http://localhost:8080/webapps/servlet/HelloWorldExample

(Again, HelloWorldExample is to replaced by the name of the servlet to be used in the application).

4.4 Connecting Midlets to MySQL

To connect a MIDlet to MySQL, inside the MIDlet code a HttpConnection must be opened. Create an instance of such a connection, HttpConnection http = null. An InputStream is also used to read information from the server, InputStream iStrm = null. After doing this, code such as the following will open a connection to the servlet.

```java
try {
    http = (HttpConnection) Connector.open(url-of-servlet);
    http.setRequestMethod(HttpConnection.GET);
    iStrm = http.openInputStream();
    Boolean ret=processServerResponse(http,iStrm);
}
```

The code above will access the server using the GET method (see Section 4.2.2). The processServerResponse method will be used to process the response of the server. The servlet will read the sentence to be translated, and
send the resulting translation back to the MIDlet. `processServerResponse` will read this information from the server, format it and print to the screen of the mobile phone. See Appendix A for more information.

4.5 Conclusion

In this chapter the necessary connectors for enabling communication between MIDlet files and the MySQL database were described. The notion of a JDBC driver, a series of java classes used to allow java to interact with a database server, was discussed. It was explained that the JDBC driver alone is not sufficient for the purposes of this project and that the use of servlets was also required.

The principle points of servlet technology were described and brief description of the toolkits used for developing servlets was provided. These two toolkits are the JSWDK and Tomcat. A description was then given of the way in which servlet technology can be used to enable the required MIDlet/Database communication.
Chapter 5

Populating the Database, a 2nd Approach
5.1 Outline

Imagine a tourist is about to go abroad. As he will only be in the foreign country for a few nights it seems like a waste of money to buy a whole phrase book when all he will need is a few simple phrases. Wouldn’t it be handy to simply look up the few phrases required on the Internet before he or she leaves, input them into a database system stored somewhere on line and then be able to access this database via his mobile phone when in the foreign country? This would be especially useful if visiting a country where a relatively obscure language is spoken, or if the phrases needed were somewhat specialised (related to a field of work perhaps) and thus not likely to be available in traditional phrase books.

The next section describes the implementation of such a system.

5.2 Populating the Database via Webpage

A webpage was set up in which the user can input the phrases they will later require when abroad. The webpage contains a FORM to be completed by the user. They are asked to input the source language, the target language and the sentence/translation pairs they wish to enter. This information will then be inputted into the database.

5.3 FORMs

A HTML form is a piece of code embedded inside a HTML document that contains interactive features such as textboxes, radio buttons, menus, etc. A user can enter values into this form by modifying its controls. They can enter text into textboxes, click on radio buttons, etc. Usually a form is accompanied with a Submit and a Reset button. The user can thus change the details they have entered into the form before it is submitted. The general layout of a form is:

```html
<FORM action="http://someaddress.com/program" method="post">
  <LABEL for="someName">someName: </LABEL>
  <INPUT type="someInputType" name="someName">
```
5.4 Processing FORMs

Data submitted in a form is processed in one of two ways:

1. Using PHP
2. Using Servlets

5.4.1 PHP

When using PHP to process the data entered into a form, the name of the PHP file is used as the action value in the FORM tag. There is a global array called $_POST['variable name'] which is built into PHP. This array contains all the information posted to the PHP file via the form, where 'variable name' is the name value of the input tag in the FORM:

```php
<?php
$variable name = $_POST['variable name'];
Do some action with the array
?
```

Because of the type of data being processed in this project, it is necessary that a tool that can access MySQL is used. PHP allows such a connection, and is commonly used as a web interface for MySQL databases. To connect to the database using PHP, the following code is used:

```php
<? $result=mysql_connect("addressOfDatabase","loginName","loginPassword");?
```
This code must be placed before all other code in the PHP file.

The database can be queried by adding the following to the PHP code:

```php
<? $result=mysql_db_query("databaseName","SQL query"); ?>
```

### 5.4.2 Servlets

Servlets have already been discussed in Chapter 4. What follows is a brief description about the way in which servlets interact with HTML forms. The servlet processes data from the form by carrying out a doPost() method. The servlet takes in possible parameters from the form. These are identified by the `name` value in the `INPUT` tags, e.g.

```java
protected void doPost(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
    String s1 = req.getParameter("variable_name");
```

(presuming the value contained in the “variable_name” field is of type text). This data, now easily accessible in String form can then be entered into the database by creating a Statement object and using it to execute an UPDATE to the table.

Although both methods of form interaction are viable, a java servlet is used in this implementation to enter the user’s phrase/translation pairs into the database. The servlet was chosen over the PHP simply because servlets were used for previous interaction between the database and the mobile phone.

The servlet takes in the values entered by the user, checks for special characters and turns them into their equivalents code (see Section 3.3). It then checks to see if the sentence they wish to translate is already in the database. If this was not checked, the phrase will be entered again into the database, with a new `id` number. By checking to see if the phrase is already stored in the database an UPDATE can be executed, instead of an INSERT statement. Otherwise, a simple INSERT is executed.
Sample interface for phone translation program/database

Please enter source and target language below (in small caps):

1st Language: 

2nd Language: 

Enter sentences in source language followed by a ':' followed by the corresponding sentence in the target language. Follow your input with the string "END:"

e.g. hello:bonjour:

Hello:Bonjour:
END:

Figure 5.1: Form Input Webpage
5.5 Conclusion

In this chapter a system for database population via the Internet was proposed and described. A description of HTML forms and their features and capabilities was given, along with a brief description of the form syntax. The way in which these forms can be processed using either PHP or servlets was also described, and a screenshot of the resulting HTML webpage containing the form used in this project was presented.
Chapter 6

Improving Translation Quality
6.1 Improving Quality

6.1.1 Frequency and Success

Once the basic system was in place, to increase the functionality of the system, a counter was put in place to keep track of the frequency with which a certain translation is made. Thus, every time a request for a translation is made the frequency counter for this sentence is incremented. This takes into account the source and target languages for the translation. For example, if a request for a translation into Spanish of the English sentence “Where are the toilets?” was made, the freq\textunderscore english\textunderscore spanish column is automatically incremented. In this way, one can keep track of the translations that are most in demand. For every language pair there is a frequency column associated with it. Thus, French sentences have freq\textunderscore french\textunderscore english, freq\textunderscore french\textunderscore spanish, freq\textunderscore french\textunderscore german, freq\textunderscore french\textunderscore italian and so on.

While this is an interesting thing to keep track of, it is not very useful. To make use of this frequency counter, another serious of columns was added to the translation table that rates the success of a translation, again based on language pair involved. Sometimes a direct translation is simply not useful. If a person was to ask “Can you open a window in here?”, sometimes all they are trying to express is “Phew, it’s hot in here”, etc. The system attempts to deal with this by asking the user after every translation whether or not he/she would like to see another translation of their input. If they choose not to avail of this option, then the translation is deemed successful. For example, in the above translation of “Where are the toilets?”, the success\textunderscore english\textunderscore spanish counter would be incremented. If, however, the user chooses to look at another translation, then the success counter is decremented. The translation has not met the needs of the user.

There are two main advantages to such a system:

- Firstly, if there is more than one possible translation of a given sentence, the system can keep track of which translation is most successful and it is this one that is returned to the user.
- Secondly, if only one possible translation exists, we can compare the
frequency counter and the success counter to see whether or not a translation is a good one. Obviously if a translation is carried out 20 times but only has a low success rating (e.g. of 3 or 4) then this translation is not a good one and a better translation should be made available.

There is, however, one main problem with the system. The system is based on the idea of having two or more translation for one thing, and keeping track of which of these translations is better. However, it is unlikely that there would be two or more translations for one thing stored in the database, unless they are inputted by the administrator. When inputting via the webpage, if a sentence to be translated is already in the database, the program simply overwrites the previous translation that was there with a new one. The system works in this way to accommodate the addition of new languages. When a sentence is inputted via the webpage, if it is already in the database, the program assumes that the user is adding in a translation for the sentence in question in a new language and performs an UPDATE instead of an INSERT (see Section 5.4.2). Thus, realistically, there aren’t many entries in the database where there are multiple possible translations.

6.1.2 Partial Matching

An application that will only return a translation if the user enters a sentence that matches its equivalent string in the database exactly is not very useful. Take for example a situation where the user wishes to know how to get to the cinema. The entry in the database matching such an inquiry is “Where is the nearest cinema?”. If the user happens to enter this exact string then a translation would be returned. However, if the user asks “Where is the cinema?” they will be told that no translation could be found. To make the translation server more useful, a partial matching system was implemented so that the system no longer required the user to enter the exact string that is stored in the database.

Approximate String Matching

Approximate String Matching is a process whereby substrings of a text string close to a given pattern string are found. Graham Stephen (Stephen, 1992) explains that the approximate string matching problem can be stated as:
Given a pattern string $x$, with $|x|=m$, text string $y$, with $|y|=n$, where $m,n>0$ and $m\geq n$, an integer $k\geq 0$ and a distance function $d$, find all the substrings, of $y$ such that $d(x,s)\leq k$.

Thus, the aim of approximate string matching is to find all the substrings of a string that have a distance of $k$ or less from a given pattern, for some given distance function, $d$. One must then decide on this distance, $k$. How close to the original string does the new string have to be for it to be considered a partial match?

Partial matching can be based on character count or word count. One can decide that a string entered by the user is a partial match to a string in the database if it is no more than 2 characters out. This would deal with plural problems. (e.g. cinema VS cinemas), but would lead to problem when sentences such as “Where can I buy some tea” could be translated as “Where can I buy some ties”, etc. Thus, partial matching based on characters is not particularly useful.

The partial matching system implemented in this project allows for a partial matching of strings that have two or less words (tokens) that differ from the original entry in the database. Testing on changing this margin of error has shown this approach to give the best results. However, like partial matching based on matching a certain number of characters, not all partial matches allowed by the system are appropriate to the user’s inquiry. Thus, when partial matching is used to return a translation the user is informed of this, so that they can be aware that the translation is not an exact translation of the sentence they inputted. Because the partial match may have a margin of error of either 1 word or 2, the user will also be informed of the percentage difference between the string they entered and the partial match used to provide a translation. This is implemented using a system where there are two methods for partial matching, $\text{partialMatch1}$ (if only one word off) and $\text{partialMatch2}$ (if two words off). If no direct translation is available and partial matching must be used, the user will first be offered the results of $\text{partialMatch1}$ and told that this match is a certain percentage off from the string they entered. If $\text{partialMatch1}$ does not return a partial match or the user is dissatisfied with this response, $\text{partialMatch2}$ will be used. Again the
user will be informed that this is a partial match and will be notified about the percentage difference between the input and output. Because the system keeps track of customer satisfaction, it is easy to check to see how efficient the partial matching is. Obviously, if the user is always dissatisfied with the partial matching results, then the system must be improved.

**Problems with Implementation**

Basing the margin or error around whole tokens and not single characters doesn’t negate the need for a method for dealing with plurals. Thus, the system must not only cater for partial matches that are one of two tokens different from the original string, but most also allow for matching of tokens and their plural equivalents. Imagine the database has the entry “Is there a hotel here?” and all of its associated translations. Were the user to input “Are there hotels here?”, which is approximately the same question, then the partial matching system would deal with this and successful match the two strings. This however is simply because it is recognising “there” and “here”, and discounting “are” and “hotels”. It is just by chance that we have returned the proper solution. However were we to enter the string:

“Where is the toilet?”

(where the corresponding entry in the database is “Where are the toilets?”), we get the response:

“Where is the police station?”

This is because it has not been stipulated that the number of tokens in the string and its appropriate partial match should be the same. Doing so, however, does not remedy the problem. A constraint was implemented to ensure that `entryString.countTokens` must equal `partialMatch.countTokens`. Once again:

“Where is the toilet?”

was entered. The system will now discount “Where is the police station?” as a match, but instead will return:

“Where is the lift?”
because this entry in the database preceeds the “Where are the toilets?” entry.

Thus the system is going to encouter problems with all “Where is the ....” questions.

**Putting Emphasis on “Key” Word**

The process implemented to provide a solution to this problem was to place a special emphasis on the most important word in the sentence. If it is ensured that this “key” word is present in the partial match string then the solution is more likely to be an appropriate response to the user’s request.

The problem of identifying this “key” word now remains.

Imagine it to be the last word in the sentence. If the clause ensuring the number of tokens in each string must be the same is removed, and if it is stipulated that not only must a partial match differ from the input string by less than two tokens, but that the last token in each string must be equal (taking plurals into account), then when the string:

“Where is the toilet?”

is entered, the desired response will be returned, i.e.:

“Where are the toilets?”

But then, what about the string:

“Are there hotels here?”

Remembering that the database entry is “Is there a hotel here?”, the desired response will again be given, but only because the last token in each string, i.e. “here”, is the same. But “here” is not the key word of the string, “hotel” is.

For example the system would fail to recognise the similarity between:
“The toilets, where are they?”

and:

“The toilet, where is it?”

In stipulating that the longest word in the string is the key word, similarly “good” results are obtained. But, again, the longest word is not always the key word and many of these “good” results are simply returned by chance. It does, however, allow matches between sentences such as:

“Where is the men’s toilet?”

and:

“Where is the toilet?”

Plurals

To look at the plural problem in more detail, the “Where are the toilets?” example will again be used. Originally, if the user was to enter the string:

“Where is the toilets?”

then the system had no problem matching this with:

“Where are the toilets?”

But, if the following:

“Where is the toilet?”

was entered, no match could be made.

To allow for plural equivalents, therefore, more than simply ”s” (or similar plural inflections) must be taken into account, but also the possibility of ”s?” at the end of a token, as the tokenizer takes “toilets?” to be a single token.

The possibility that two characters at the end of the string were different was allowed, but was one to again ask:
“Where are the toilets?”

a translation still could not be provided, because the system ends up comparing:

“Where is the toilet?” with “Where are the toilet”

The second string here does not end with a question mark.

Because of this, the partial matching system had to be improved to include a check for question marks at the end of phrases.

Even with these improvements, the matching system is not always completely successful. This leaves a potential field which could be expanded as a future project.

6.2 Improving Speed

6.2.1 Caching

There is no limit on the amount of entries that can be added to the database, but there are certain entries that will be accessed more frequently than others. Everyday queries such as “Where are the toilets?” or “Where is the train station?” will undoubtedly be asked more often than database entries such as “What is your favourite football team?” It makes sense that some sort of caching system is implemented so that data that is frequently accessed is always on hand. The main purpose of caching is to accelerate the speed of the program. Caching allows you to do tasks more rapidly. Because a system that keeps track of the frequency with which a query is submitted had already been implemented, caching was a fairly easy system to put into place.

The most_freq table

The most_freq table contains the id numbers of the twenty translations that occur most frequently. The primary key of the relation is a column that holds positions 1-20 (see Figure 6.1 below). Every translation transaction that is carried out has the potential to change the entries in this table,
because with every translation transaction the frequency value associated with a particular entry in the database will increase. Therefore, when a translation occurs there must be some system in place that checks to see if the frequency of the translation is higher than that of the last entry in the most_freq table. The next section explores possible implementations of such a constraint.

![Figure 6.1: Sample from most_freq table. pos represents frequency, with entry 1 representing the most frequent translation, 2 the next most frequent, etc. id identifies the translation row in question](image)

### 6.2.2 Triggers and Assertions in MySQL

Integrity constraints such as triggers and assertions are used in relational database to ensure that certain conditions are satisfied.

#### Assertions

An assertion is a predicate that expresses a condition that the database should always satisfy. Once an assertion has been made, the system tests it for validity. It is tested again every time the database is updated. An assertion
takes the form:

```
CREATE ASSERTION 'assertionName' check 'somePredicate'
```

After a translation has been carried out, assertions can be used to ensure that that the frequency value associated with this translation is not more than that of the last entry in the `freq` table, however if the frequency value is higher, an assertion will not enable replacing the last entry in the table with that of the now more frequent one.

**Triggers**

A trigger is similar to an assertion. It is a piece of code that not only checks a certain condition, but automatically implements a piece of code if this condition is met by the database. Thus, a trigger is a statement that is executed automatically by the system as a side effect of a modification to the database (Barry & Bryson, 2005).

Therefore, a trigger can be implemented that checks, once a translation has been carried out, whether or not the frequency value associated with this translation is higher than that of the last entry in the `freq` table and carry out the required update to the `freq` table if it is:

```
CREATE TRIGGER check_freq
AFTER UPDATE ON tr
FOR EACH ROW
    update freq.freq if necessary;
end;
/
show errors
```

### 6.2.3 Java: Servlets Integrity Constraints

Another way of checking and updating the `most_freq` table is to extend the code in the servlet that carries out the translation. This is done by executing a query that returns the last entry in the table, checking the frequency value of the row associated with the `id` number returned, comparing this value to the frequency of the current translation, and executing an update if necessary.
6.3 Conclusion

In this chapter, the advantages of keeping track of the frequency with which a particular translation is carried out, and of keeping track of the success of the translation, were described. The way in which this was implemented in the project was also described. A description of the theory behind approximate string matching was given, along with the way in which approximate matching was used in the project, and the problems encountered in using the partial matching system. An account was given of the most.freq table and the way in which it is used to enable caching. Finally, it was shown how triggers or servlets can be used to allow necessary updates to this frequency table.
Chapter 7

Conclusions and Future Work
CHAPTER 7. CONCLUSIONS AND FUTURE WORK

7.1 Summary

This thesis has successfully implemented a system for a translation server that can be used on a mobile device. The required tools have been used and communication between the mobile device and the online database server has been put into place. A system that allows clients to enter data into the database via the Internet has also been implemented.

Chapter 2 has explored all the steps necessary to create and compile MIDlet code. The chapter highlighted the functionality of the Java Wireless Toolkit and the way in which it facilitates the creation of MIDlet suites.

Chapter 3 explained the choice of database server and provided a description of the way in which the database was set up and manipulated to serve the purposes of the translation server. It also described the problems encountered, highlighting the charset problems, and provided solutions to these problems.

Chapter 4 discussed java servlets in detail. Servlets are behind all the communication between the different constituents of the project and were thus very important in the implementation of the translation server. Toolkits that facilitate the building, compilation and testing of servlets were described and the way in which the MIDlet code communicates with the servlet was also described.

Chapter 5 provided a description of the webpage that is used to allow users to enter data into the database. The code behind the webpage was described, along with the way in which the data inputted into the webpage is processed and entered in the database.

Finally, Chapter 6 described the way in which the system keeps track of the frequency and success ratings associated with a particular translation and uses these ratings to improve the quality of translation returned to the user. It also described the implementation of a partial matching system and the problems encountered, as well as describing the implementation of a caching system to improve the overall speed of the program.
7.2 Achievements

As stated in section 1.3, the aim of the project was to set up an interactive “phrase book” to be used on a mobile phone. I feel that this has been successfully implemented. The translation server translates phrases to and from English, French, Spanish, German and Italian and the webpage facilitates the adding of new languages with ease. The partial matching system also allows approximate matching of translation requests with entries in the database.

While the system is in no means perfect, I feel that, taking into consideration the time constraints on this project, I have achieved a lot and implemented the system to the best of my abilities. The project has been rewarding because it has shown me that undertaking and successfully implementing such a project is possible, and has allowed me an opportunity to use the knowledge gained over the last four years of my undergraduate career.

7.2.1 Problems Encountered

Most of the problems I encountered in undertaking this project were not connected to the implementation of the project itself, but more with my lack of knowledge in the mobile communication area. I found that simple tasks such as setting my classpaths took up a lot of my time, simply because I did not know what I was doing. Thus, tasks that should have been done in minutes, ended up taking me quite awhile. My lack of knowledge vis-a-vis servlet technology also held up the implementation process. I spent time trying to do things you simply can’t do with servlets, and code that I could write in seconds at the later stages of the project took a long time to write at first.

In terms of the approach taken in this project, the main disadvantage is the inflexibility of the sentences stored in the database. Because phrases have been stored in the database, and not simply words, manipulating the database entries is difficult. It also means, that the user is required to enter either the exact database entry, or something very similar to it, to obtain any kind of decent response from the system.
7.3 Future Work

A translation server, such as the one described in this thesis, can always be improved upon. In accordance with this, the following section, describes some directives for future work.

7.3.1 Incorporating “TextSpeak”

People simply do not have the patience for tapping on a telephone keypad as they would for a computer keyboard. When texting, most people tend to use “textspeak”, for example, using “2moro” instead of “tomorrow”. As it stands, the system has no way of dealing with such input.

7.3.2 Improving Flexibility

The partial matching system implemented is just not flexible enough to able to properly deal with translation requests, and needs to be improved dramatically. One approach would be to tag the input to the database so as to be able to identify properly the “key” word in a sentence. This would greatly improve the approximate matching system already in place.

7.3.3 Automata

As this is a translation project it would seem logical/efficient to set up the system as a finite state automaton of some kind, or to at least have some sort of grammar rules generating the responses to queries. One possible directive for future work would be to change the database structure to replicate an automaton. Such an approach would also facilitate a better partial matching system, but would lead to greater problems when relating the languages to each other and when inputted data to the system.

7.4 Concluding Remarks

The mobile phone industry is growing all the time. There is great scope for work in this field and the demand for mobile phone services (as explained in Section 1.2) is huge. The implementation of a translation server is but one
small area of work within this field, but is a service that is of use to everyone in a world where travel is becoming more and more common place.
Bibliography

Java archive (jar) files.
http://java.sun.com/j2se/1.5.0/docs/guide/jar.

Midlet attributes.

Servlet essentials.


User guide: j2me wireless toolkit.


Day, B. Develop wireless applications using j2me.
http://www.javaworld.com/javaworld/javaone00/j1-00-j2me.html.


Mahmoud, Q. (2001). Midp inter-communication with cgi and servlets.  


http://www.puca.ie/041210%2520Puca%2520Goverment%2520Survey%2520Release%2520(int%2520ver).doc.


Appendix A

Code

See source code on accompanying cd
Appendix B

Example of Partial Matching

The following is an example of a translation emulation involving partial matching:
APPENDIX B. EXAMPLE OF PARTIAL MATCHING

Figure B.1: Partial Matching 1
Figure B.2: Partial Matching 2
Appendix C

Tools Utilised

EasyPHP Installation

Download EasyPHP from http://www.easyphp.org\(^1\). Double click on the downloaded file and select the target directory. Follow the instructions that appear.

Setting up MySQL on macneill

Email the Computer Science Department Helpdesk, help@cs.tcd.ie, to set up a MySQL account on macneill. They will provide you with a user name (your college login), password and database account.

SSH into macneill.cs.tcd.ie, using your college login and college password. To access MySQL, type `mysql -p nameOfDatabaseAccount`. You will then be prompted for a password.

\(^1\)Last verified: April 2005.
Complete Code for Special Characters

<table>
<thead>
<tr>
<th>Character</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>à</td>
<td>a1</td>
</tr>
<tr>
<td>â</td>
<td>a2</td>
</tr>
<tr>
<td>á</td>
<td>a3</td>
</tr>
<tr>
<td>è</td>
<td>e1</td>
</tr>
<tr>
<td>ê</td>
<td>e2</td>
</tr>
<tr>
<td>á</td>
<td>e3</td>
</tr>
<tr>
<td>ç</td>
<td>c1</td>
</tr>
<tr>
<td>Ç</td>
<td>C2</td>
</tr>
<tr>
<td>ó</td>
<td>o1</td>
</tr>
<tr>
<td>ô</td>
<td>o2</td>
</tr>
<tr>
<td>ó</td>
<td>o3</td>
</tr>
<tr>
<td>ì</td>
<td>i1</td>
</tr>
<tr>
<td>ì</td>
<td>i2</td>
</tr>
<tr>
<td>ù</td>
<td>u1</td>
</tr>
<tr>
<td>û</td>
<td>u2</td>
</tr>
<tr>
<td>ú</td>
<td>u3</td>
</tr>
<tr>
<td>ü</td>
<td>u4</td>
</tr>
<tr>
<td>ñ</td>
<td>n1</td>
</tr>
<tr>
<td>¿</td>
<td>?1</td>
</tr>
<tr>
<td>¡</td>
<td>!1</td>
</tr>
</tbody>
</table>

Table C.1: Code for special characters