MASSEY UNIVERSITY

INSTITUTE OF INFORMATION AND SCIENCES TECHNOLOGY

INTERACTIVE TRANSLATION OF JAPANESE TO KOREAN VIA CELLULAR TECHNOLOGY

STEVE MANION

2006

SUPERVISOR
LIYANAGE DE SILVA
AMAL PUNCHIHEWA

Contents Page

Summary

Translation of language via cellular technology is now in the crosshairs of cellular phone developers. Still yet to be achieved is translation of Japanese to Korean. The two languages share several aspects which makes translation between them a key candidate for the translation algorithm I designed, abbreviated as FBCT, *Fish Bone Composition Translation*. Typical translators attempt to translate sentences using a combination of dictionary attacks, probability techniques or segmented translation which is usually offers a piecewise solution of sub translations concatenated together. These methods work to an extent but not enough importance is weighted on the actual pattern of the sentence or how the translated segments are concatenated back together to provide the final translated sentence. In reality conventional methods can be successful, but usually rely on situational databases of words and grammar depending on the scenario. For more broad translation of text these methods can fall short and we receive garble translations because the overall meaning or intention of the sentence can be lost during translation. FBCT is an algorithm I designed which treats every possible sentence as a unique combination of grammar and words when it translates, so there is an applicable blueprint and translation function designed in advance for every possible sentence. By coordinating and taking advantage of the processing power of a server and portability of a cellular phone we have the ideal hand held translator. It can be constantly updated at the server end as language evolves, and access rights to the translation service are easily loaded on to the phone account of any visitor from Japan to Korea at the airport terminal.

1 Introduction

1.1 The Service

The service will provide customers with the ability to translate anything they wish to express using a cellular phone that they can easily acquire from the airport as the enter Korea. Visitors from Japan to Korea can have piece of mind that they will have the ability if caught out in any difficult situation, to clearly explain their way out of it. The service also converts the cellular phone to an educational device, as users will have the ability to learn simple expressions and the fundamentals of the grammar used in previous translations stored within the phone, as well as other options such as synthesised speech files of the translated message for the user to play back. The service is always up to date at no extra cost for its users. Any new words or grammar constructs that come about between the translations of the two languages can be compensated for and updated at the server end; this means no wasted time on update downloads or fees for an updated service. Lastly the service is very transferable, a simple download onto your phone to give you access rights to the service is all that is required. If the user wishes to no longer use the service their subscription is easily cancelled and the software itself the user can delete from their phone. The operations of this service and how it is to be implemented will be covered in detail within this report.

1.2 The Market

Japan is Korea's number one source as a country for tourism revenue, with Japanese visitors to Korea far exceeding the amount of visitors from any other country. Therefore, tourism related products should in particular be geared for the Japanese market. This service forges together the two industries of IT and tourism together, which are two of the three top growth industries expected to be in the next 20 years. A survey conducted in this research, showed very positive results indicating that most Japanese tourists would be interested in having access to such a translation service whilst they are in Korea. Even if only a fraction of them actually were willing to acquire the service whilst in Korea, the direct and indirect profit that can be accumulated by Korea is substantial. A Market Analysis based on a conducted survey has been put together to demonstrate the value of this project which can be later viewed in this report.

2 Background

2.1 Previous Research

Oriented to the development of practical SST system for the Olympic Games, ATR-SLT, ETRI of Korea, and CAS-IA jointly signed the CJK (Chinese-Japanese-Korean) cooperation agreement. The purposes of the agreement is to joint research and develop a Speech-to-Speech Translator (SST) system, which converts spoken expressions, related to travel conversation between the three languages (the Japanese language, the Chinese language, the Korean language). [1]

2.1.1 ATR Japan

The Advanced Telecommunications Research Institute International (ATR) of Japan was established in 1986 and was the first laboratory to work on the speech-to-speech translation in Asia. Between 1993 and 1999 ATR designed various prototype systems which could translate spoken Japanese into Korean, German, English and Chinese. In 1998, ATR successfully developed the ATR-MATRIX Japanese-English bi-directional speech translation system, which was later operational on PC notebooks and accessible for users with a cellular phone. This of course was one of the first cellular phone based translators to be developed. [1] ATR now strives to design translation systems which focus on real world situations, comprising of situation based translations in which particular grammar and utterances occur frequently.

2.1.2 ETRI Korea

The Electronics & Telecommunications Research Institute (ETRI) of Korea has been conducting research on speech translation under government funding. The main focus of ETRI is on how to use speech translation technology in real life. This is because the feasibility of the technology that had been shown in previous demonstrations, such as the ETRI-KT-KDD demonstration on hotel reservations in 1995 and the C-STAR II demonstration on travel planning in 1999. These prototype translators could translate speech related to travel planning with a 5,000 word vocabulary from Korean to English and from Korean to Japanese. [2] Due to their success in situational translation systems, ETRI has targeted on mobile phone speech recognition, translation of simple but crucial expressions for travellers, and dialogue style speech synthesis. [1]

2.2 Related Products

Translation Software is appearing on all portable platforms as people continue to travel around the world and carry these devices; it makes the spread of such products quite easy.

2.2.1 SK Telecom's "Real World Phone"

On the 13th of September, SK Telecom and Samsung Electronics disclosed to the public that it would release the SCH-V920. The importance of this phone is its ability to be used in both CDMA and GSM networks, allowing the phone to be used almost anywhere in the world. The phone also comes with a translator built in. "An *Anycall Translator* function has also been included, allowing translation of simple expressions in various situations into English, Japanese and Chinese in airports, hotels and stores." [3] This product is able to translate in the opposing direction from Korean to Japanese at an elementary level. The SCH-V920 is a flagship phone that has paraded onto the development scene of translation via cellular technology; more are expected to follow.

Fig 1. The new SCH-V920 Real World Phone [3]

2.2.2 Sony PSP's Talkman

Earlier this year Sony released the game "Talkman" for its PSP (Play Station Portable) which was described by GameSpot, a leading gamer magazine as an "Interactive translator and language tutor that helps you communicate in six foreign languages, and makes for a novel, if flawed, travelling companion." [4] This device takes translation from a more educational angle making it into a game to encourage users to learn the language whilst on their travels. However as a real portable translator it has been reviewed by many, including GameSpot as falling short of the mark. Mainly due to the games limited phrases in situations and loading time for the correct scenarios, defeating the translator's ability to perform in real time for the user.

Fig 2. A screen shot of the Talkman translation game [4]

2.3 Conventional Translation Methods

2.3.1 Dictionary-Based Machine Translation

This method of translation is at the very atomic level, with no overhead complexities; words in a sentence are replaced word for word from the source language to the target language. This low level of translation almost always generates meaningless strings of words; however for very short text translations it can be adequate.

2.3.2 Statistical Machine Translation

This technique is derived from information theory. "Essentially, the document is translated on the probability that a string in the Source Language A is the translation of a string in the Target language B using parameter estimation." [5] This method can be highly effective in controlled scenarios, for instance booking a train ticket where probability of certain statements and words are very high.

2.3.3 Example-Based Machine Translation

This is not a solid technique for general translation; however it is a nice add-on that can improve the performance of a translator. Basically it generates rules about which words can be swapped in a sentence without changing the overall meaning of a sentence. More or less, it gives the translator the ability to paraphrase its output into the target language if needs be.

2.3.4 Interlingual Machine Translation

This is where an intermediate that is language independent is created by analysing the source language in depth, then using the intermediate to reconstruct the equivalent meaning in another language of the user's choice. This method is very rule based and when there are numerous languages at play, the process becomes very difficult as the intermediate must cater for more languages, in turn becoming more diverse but losing dexterity to achieve complex translations specific to each respective language. The method of translation used in this project is a variation of this technique; defining an intermediate labelled the Fish Bone Composition that is dedicated to taking advantage of the similarities the Japanese and Korean language. There are only two languages involved enabling it to be highly accurate as it can deal with a dynamic range of complexities between the two languages alone. More on this will be explained later.

2.4 The Japanese & Korean Language

Before one can fully appreciate *Fish Bone Composition Translation*, one must understand some key fundamentals and similarities that occur between the two languages to understand why they are key candidates for bidirectional translation.

2.4.1 Particles

European languages usually rely on word order and the insertion of appropriate prepositions to define context of each word in a sentence. On the contrary, the Japanese and Korean language share what are known as *particles*. A *particle* is placed behind each word or group of words forming a sub clause to define its context in the sentence. Therefore the word order can often be moved around without affecting the overall meaning of the sentence. Detection of the existing *particles* and the word types they appear after in a sentence is one of the key ways my translator is able to identify the grammar constructs that exist in a sentence. There are *particles* that exist in the Japanese and Korean language that follow almost the same behaviour within the two languages, a key aspect which should be taken advantage of during translation.

2.4.2 Structure

The Japanese and Korean language share very similar grammar constructs, such as verbs and adjective always appearing at the end of a sentence, and sentence characteristics such as the subject sometimes being omitted and so forth. A common problem when translating for a European language to an Asian language or vice versa is finding grammar constructs that may exist in one language but not exist in another language. This rarely happens between Japanese and Korean due to their close relation in structure.

2.4.3 Word Derivation

Both the Japanese and Korean imported many things from Chinese culture over the centuries including the Chinese alphabet, which is still used by the Japanese today. Borrowing the Chinese alphabet, naturally Chinese words were also implemented into these two languages. Therefore there is a reasonable portion of the two languages still use these Chinese words, even though not pronounced or spelt the same; they still largely retain their original meaning which makes finding a word in Korean with the equivalent meaning in Japanese a little easier.

3 The Translator

3.1 Design

3.1.1 Data Model

The system comprises of six main class types and a database that altogether are called upon by the control class to achieve the overall translation. There are three types of main classes, known as Procedure classes, Support classes and Container classes. The database strictly holds information and interacts with the Dictionary class, and lastly the Container classes are stored and referenced to their respective registers as there are multiple instances of them in a translation.

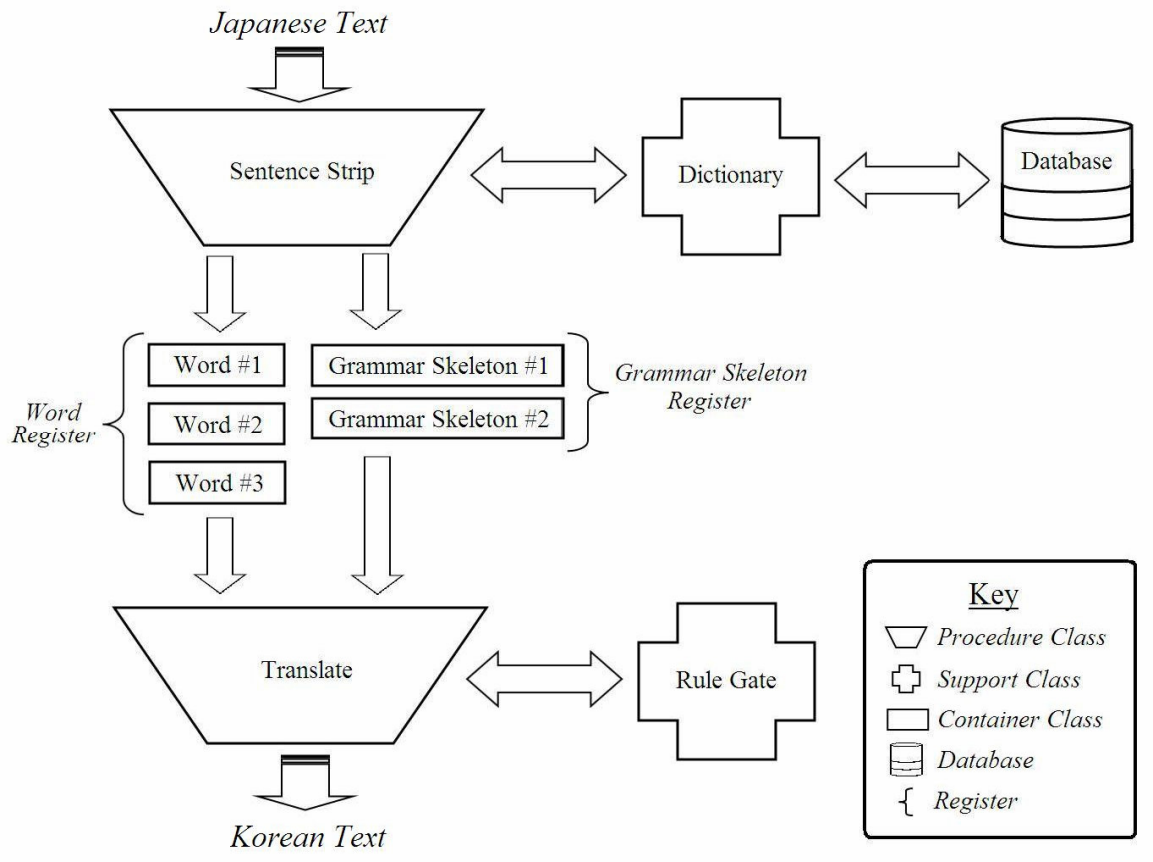

Fig 3. Class type and interaction revealed by the Data Model of the Translator

Procedure Class

The two Procedure Classes *Sentence Strip* and *Translate* do the main work within the algorithm. *Sentence Strip* is responsible for identify the blueprint grammar pattern of each sentence and surgically removing and storing required translation data in the Container Classes. Conversely, *Translate* is responsible for using the Container Classes to reconstruct the equivalent Korean sentence in respect to the original Japanese sentence, in other words providing a translation. Both of these classes perform the two major operations in a text translation, the breaking down and reconstruction of a sentence. Therefore they are known here on as Procedure Classes.

Support Class

The two Support Classes *Dictionary* and *Rule Gate* serve as aids to the Procedure Classes, providing the necessary resources to execute a text translation. *Dictionary* provides all the data required to construct the Container Classes. Its greater purpose though, is to enable the Procedure Class *Sentence Strip* to interactive with the database. *Rule Gate* provides all the grammatical conventions that are intertwined in the Korean language as methods for the Procedure Class *Translate* to use as it sees fit to execute a successful translation.

Container Class

The purpose of Container Classes *Word* and *Grammar Skeleton* is to strictly hold information, and act as a complex data variable enabling more accurate and dynamic translation that does not lose the context of the subject sentence in the process. *Word* provides detailed data about the word concerning its type, Korean equivalents and many forms of use. *Grammar Skeleton* on the other hand records all data associated with the overall sentence composition for later use.

Database

The Database naturally acts as the giant warehouse of data that is required for the translator to operate functionally.

3.1.2 Key Class Profiles

Sentence Strip

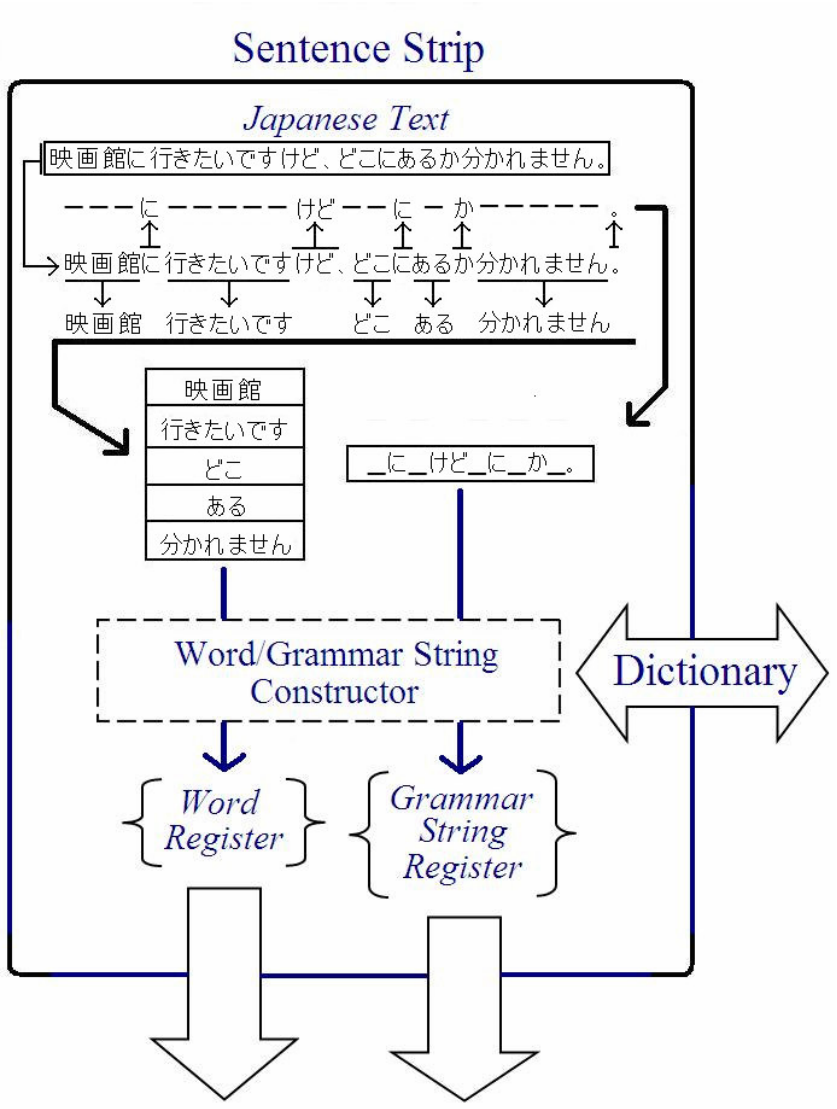

Fig 4. Internal representation of the Procedure Class 'Sentence Strip'

Firstly, the Japanese text is input as an operand in the construction of the class *Sentence Strip*. Once generated, the control class invokes *Sentence Strip* to strip each sentence in the Japanese text of all its key identifying aspects. For each word found, the Container Class Word is generated. Each sentence is traced and the Container Class *Grammar Skeleton* is generated for that particular sentence pattern. As can be seen, the *Dictionary* class interacts with *Sentence Strip* providing it with the resources it needs to construct instances of the Container Classes.

Word

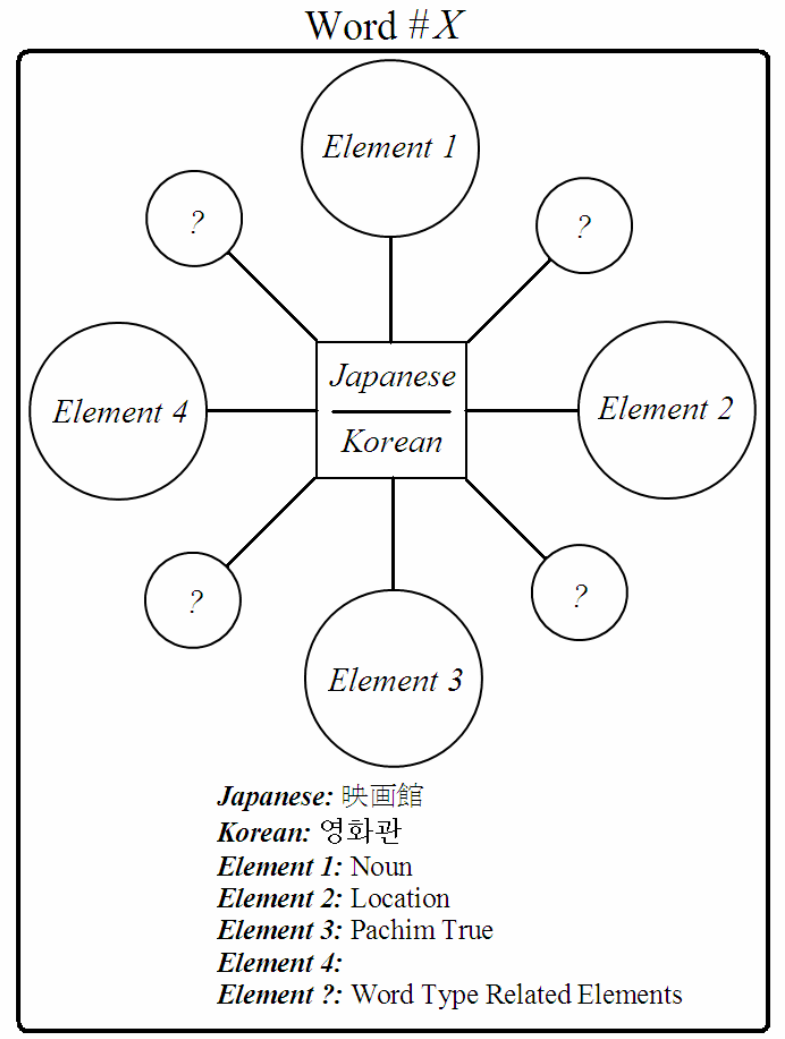

Fig 5. Internal representation of the Container Class 'Word'

For each discovered and identified word found in a sentence, the Procedure Class *Sentence Strip* generates an instance of the Container Class *Word*. Figure 5 illustrates the composition of the class *Word*. Centrally it holds the Japanese word along with its Korean equivalent(s). Then depending on the word type and description, other additional information is also collected about the word and also is stored in the constructed instance of *Word*. From our sample Japanese sentence, let's take the first word, '映画館' (Cinema), and refer to the diagram of its construction. The Korean equivalent is '영화관', its type is *Noun* and its description is *Location*. Lastly a Korean word either

has a *Pachim* or not. Explanation of what a *Pachim* is not necessary however you should know it is important as words with and without *Pachims* are treated differently in Korean Grammar. Since the word type is a *Noun*, aside from *Noun* alternatives there are few other variables to be stored as elements in the instance of Container Class *Word*. However say if the instance of Container Class *Word* was a *Verb*, there would be many elements at play, such as the *Verb Tense*, and so forth. Storing such information about a word may seem resource wasting and time consuming, however the true benefits of it will become apparent later in the explanation of this example.

Grammar Skeleton

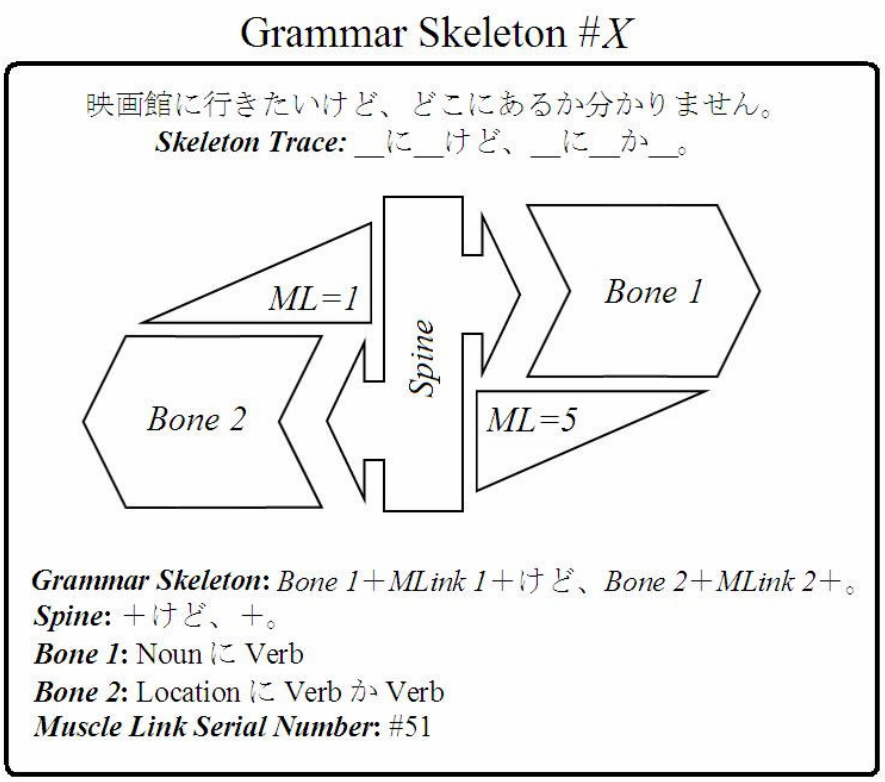

Fig 6. Internal representation of the Container Class 'Grammar Skeleton'

From figure 6, it starts to become apparent how the name *Fish Bone Composition* is an appropriate name for this method of translation. The *Grammar Skeleton* is traced, which basically means all words are stripped from the sentence, only leaving particles and grammar constructs behind. This is what we call the *Grammar Skeleton* and it comprises of three component types listed and defined over the following page.

Spine

The *Spine* of the sentence is a concatenated string of all the grammar constructs, omitting any particles that occur. It could be considered as the main sentence trace, which can link all of the other sub clauses together; therefore it is appropriate to label it as the *Grammar Spine* of the sentence. In English, words such as *but*, *because* and *when* would be caught into the Grammar Spine.

Bones

As grammar constructs are caught into the *Grammar Spine*, the groups of neighbouring omitted particles are caught into a *Bone*, which is the equivalent of a sub clause. In this sense, there lies a *Bone* between every grammar construct caught in the *Grammar Spine*.

Muscle Link Serial Number

As mentioned earlier, a lot of translators that perform segmented translation and then put the segments back together to provide the final translation usually fall short because not enough attention is paid to how the sentence is put back together. *Bones* are usually connected together by links of *Muscle*, so it makes sense to use this term when describing the process of connecting the *Bones* back to the *Grammar Spine* of the sentence. Simply put, there are several ways to alter the end of a sub clause, a *Bone*, to connect it back into the main sentence. Each number as it occurs in the MLSN, the *Muscle Link Serial Number*, references to how the next *Bone* should be linked back into the *Grammar Spine* of the sentence. How this is done can be seen later.

Translate

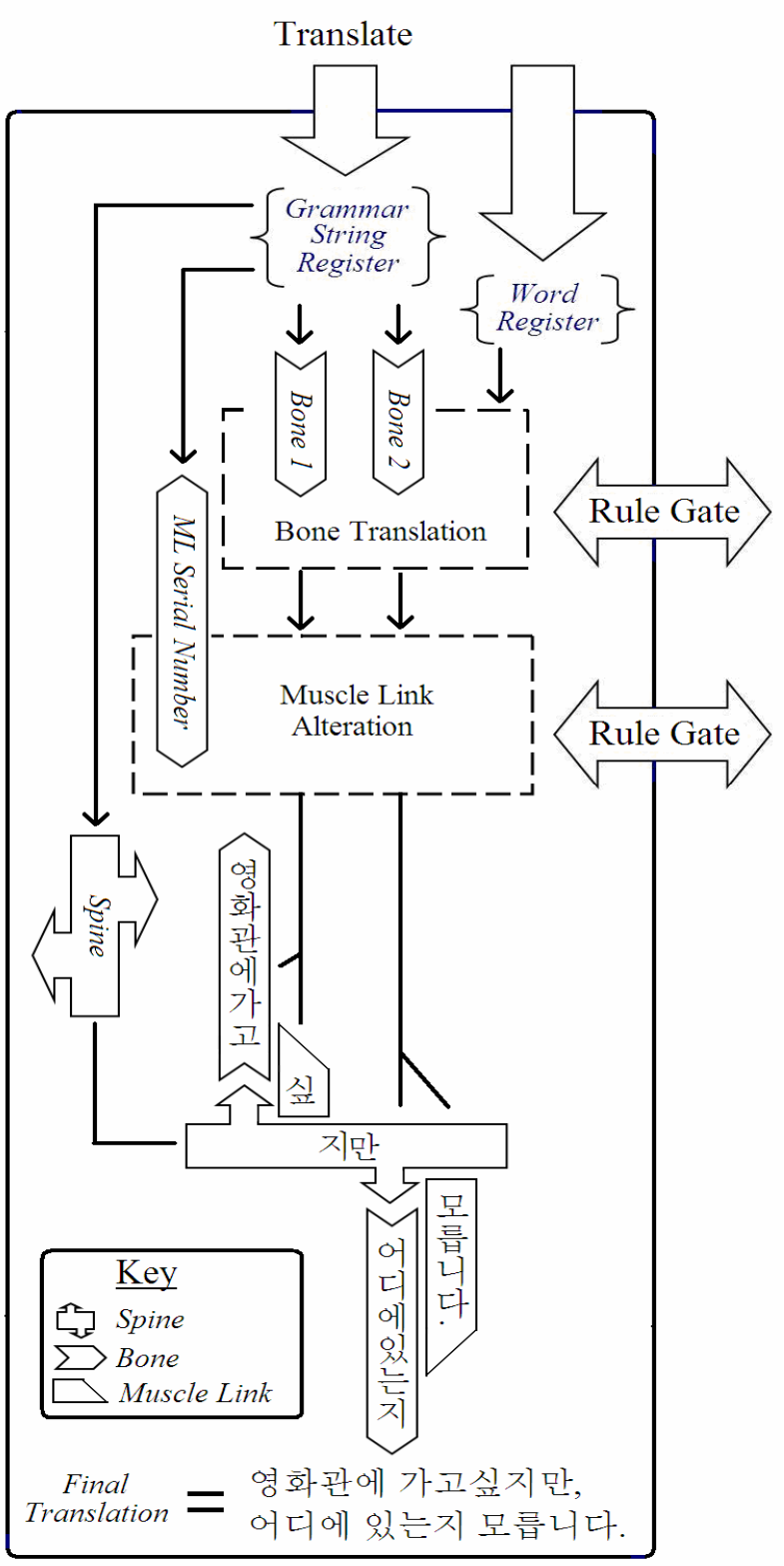

Fig 7. Internal representation of the Procedure Class 'Translate'

Once all the necessary Container Classes have been created and stored away into their respective registers, they are input as operands in generation of the Procedure Class *Translate*. Once generated, the control class requests a translation; this being the final output of the translator. First of all, each of the *Bones* is translated and *Words* are entered where appropriate to complete the translated *Bone*. The Support Class *Rule Gate* assists by providing the necessary methods required to adhere to all grammatical rules that exist within the Korean language. Once the *Bones* are translated, they now need to be altered so they can be linked into the *Grammar Spine*. This is where the *Muscle Link Serial Number* (MLSN) comes into play. If we take a look at our example the MLSN is #51.

The number 5 in the MLSN calls for a "Verb/Adjective Stem Reduction" *Muscle Link*, so for the first *Bone* in the sentence, the Verb form '싶습니다' (to want to ~) is reduced to its verb stem '싶'. This now becomes the first *Muscle Link*. Following this, the number 1 calls for a "Sentence End" *Muscle Link*, so the second *Bone* in the sentence has its final *Word* ended normally with a period concatenated to it, '모릅니다.' (to not know), this indicates the end of the sentence. Each sentence having to end at some point, we can assume that every MLSN will end with the number 1 (#???1). The *Grammar Spine* of the sentence is what is used to generate the MLSN, as the only reason the last word in a *Bone* should require changing is because it has come across a grammar construct, which is precisely what the Grammar Spine captures. Now look at the final process that occurs in *Translate*. The *Grammar Spine* is laid out and each respective *Bone* in the sentence is connected to the *Grammar Spine* via its respective *Muscle Link*. From figure 7, one can see how the translated sentence is reconstructed. In addition to this, table 1 below can be used for one to better understand the actual grammatical breakdown of the translated sentence.

Japanese: 映画館に行きたいですが、どこにあるか分かりません。

Korean: 영화관에가고싶지만, 어디에있는지모릅니다.

English: I would like to go to the Cinema, but I do not know where it is.

Cinema	(Location Particle)	(want) to go	(Conventional Verb Ending)	but
映画館	に	行きたい	です	が
영화관	에	가고싶	~~습니다~~	지만

where	(Location Particle)	exist	(Noun Modifying Particle)	(Question Particle)	do not know	(Period)
どこ	に	ある	(X)	か	分かりません	。
어디	에	있	는	지	모릅니다	.

Table 1. Grammatical breakdown of the translated sentence

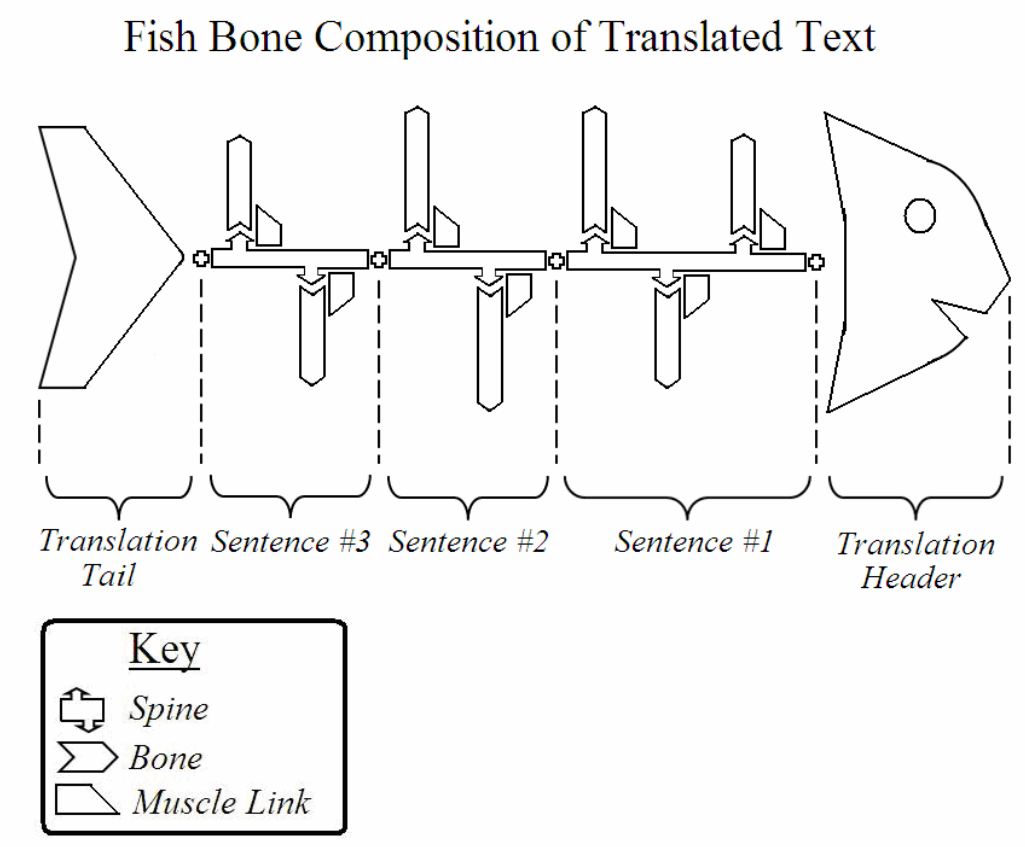

Fig 8. Representation of FBCT, Fish Bone Composition Translation

3.1.3 Fish Bone Composition

Each sentence translated and concatenated together, we begin to see an actual real skeleton, much like the type you see on a fish. Hence the term *Fish Bone Composition* Translation is conned. The diagram illustrates the *Fish Bone Composition* of a text translation. The *Translation Header* carries all information regarding the overall composition of the translated sentence and where it is destined to be sent. Typical information would be how many characters in length the message is and the subscriber's phone details so he or she can be charged and recorded using the service at that point in time. Following the *Translation Header*, is the body which consists of the overall translated text, broken down into its respective sentences and bone composition to portray the skeletal structure of a fish. Lastly, the *Translation Tail* is attached, which has the general purpose of indicating that it is the end of the translation. This is the end result of the translator and what I have now finished explaining as FBCT.

3.2 Implementation

3.2.1 Hardware

There are two important hardware features required for this project, a server and a wireless communication device such as a cellular phone. The detailed requirements of each follow:

Server

To make this project scalable, of course multiple servers are required, with the Translation application duplicated many times across the servers. Efficient use of the servers needs to be compensated by an effective routing algorithm so there is minimal inactiveness of the processor between translations for each server. Lastly the servers need to be managed effectively and updates to the Translation application should made constantly to keep up with changes that occur between the two languages to ensure customer satisfaction.

Cellular Phone

Each Cellular phone must be operational on the CDMA network in Korea. Since the project took place here in New Zealand this however was not necessary. The cellular phone used for the project in any case must have the following properties:

1. Be capable of inputting Japanese text
2. Have Korean fonts and character set installed for proper displaying of translated text messages
3. Be capable of running J2ME applications (for client server communication)

The phone chosen for this project in New Zealand was the Toshiba 803T which is an international release of a Japanese capable cellular phone from Toshiba. It is J2ME capable and translated messages can be viewed using a J2ME application known as iCJK Mail which is capable of interpreting Chinese, Japanese and Korean text.

Fig 9. The Toshiba 803T, used for this project [7]

If this project was conducted again in the future, it would definitely be more appropriate to use a phone such as Samsung's Real World Phone, the SCH-V920 which is designed for use anywhere and is automatically capable of recognizing both Japanese and Korean character sets.

3.2.2 Software

Several programming languages were used in this project. Firstly for trialling the general concepts of how the translator would work, Matlab was used as it has a nice developer environment without complex coding requirements to achieve simple tasks. Shortly after this, when the general translation concept was deemed to work, J2SE (Java 2 Standard Edition) was used. J2SE has multilingual support, and is a complex and powerful script enabling class design and the use of object orientated programming. In addition to J2SE, MySQL version 5.0 was the database of choice, as there is a lot of support available for Java scripted software that interacts with MySQL. To note for this project the JDBC (Java Database Connector) was used to interact with the database and it is also available with the MySQL package.

The translator written in J2SE, the cellular phone now needed to communicate with the server. The application written for the cellular phone to communicate with the server was scripted in J2ME (Java 2 Micro Edition). This version of Java script is a minimised version of Java that is composed together in consideration of the limited processing and memory power a cellular phone has.

The overall implementation of this project resulted in the J2ME application compiled by the Wireless Development Toolkit into a JAR file and placed onto the Toshiba 803T cellular phone. The translation application scripted in J2SE was compiled into a servlet using Apache Tomcat Server in order for it respond to any communicated requests coming from the cellular phone. To simulate the whole project functioning, a phone emulator from the Wireless Development Toolkit was used to launch the J2ME application and Apache Tomcat Server was used to act as the virtual server holding the J2SE application which responded to any requests.

Fig 10. Unicode phone emulator simulating the J2ME application (English interface)

3.2.3 Customer Access

When Japanese citizens travel to Korea, they have the option of acquiring a cellular phone at the airport that they can use in Korea that is still linked to their respective cellular phone account in Japan. They can apply for it online before they enter the country. This project, in a sense is an add-on module to this. The project will equip a Japanese person at their departing or arriving terminal to Korea with a cellular phone that is capable of Japanese to Korean translation of text. Not only will they receive a translation in Korean, they can also receive other add on services such as text and voiced pronunciation of the translation; all selectable options when they order their cellular phone. The service is there for anybody to use, the J2ME application can even be spread to other phones. However only those who have paid for the service will be able to activate it via their cellular phone. This solves the problem of the Japanese citizen still desiring to use the service without paying the due charges to their account. Customers have the option of paying per each translated text message of 80 characters, or on a weekly/monthly basis. Refer to the survey later in the results section to investigate the possible prices for this service as expected by Japanese citizens.

4 Results

4.1 Trialling of the Concept

The original algorithm was built in Matlab and was able to work at a high degree of accuracy for text translation of an intermediate level. The purpose of first building the design of the algorithm in Matlab was to trial the concept and see if it worked efficiently and accurately to develop further. Keep in mind two things regarding this Matlab trial; the algorithm only consisted of the basic idea of what the algorithm has evolved into now, Fish bone Composition was not yet established, and the grammar string involved was treated merely as a bone translation. Secondly since the program ran in Matlab, the text translation was performed in a Romanized version of the Japanese and Korean character set. Matlab was chosen for the trialling of the concept as is it a good platform for testing algorithms in their early stages. Matlab does not need extensive declaration of variables and development of classes which can be demanded by more complex programming languages which are more object-orientated.

As for the trial itself, a random sample of 20 Japanese sentences derived from a Grammar Function database of approximately 100 entries, on average 85% (17 out of 20) were able to be translated correctly. The sample sentences are generated by using reverse Grammar Functions and relevant word types place in suitable parts of the sentences at random. For example, the grammar string '__は__に__と__。' can have a variety of words placed into the gaps to generate several sample sentences of this structure. Typically these could be:

コンサート　**は**　東京　**に**　ある　**と**　思います　**。**

(I think the concert is in Tokyo.)

私　**は**　町　**に**　行く　**と**　言いました　**。**

(I said I will go to town.)

Incorrect translations usually happen because a character within a word can be mistaken as a particle or even a conjunction and added to the created Grammar String. This in turn causes one of the three mistakes which are classified accordingly below in table 1.

Test Number	Case #1: Particle Confusion	Case #2: Conjunction Confusion	Case #3: Grammar String Invalid
Test #1	1	0	1
Test #2	1	1	1
Test #3	1	0	2

Table 2. Error Classification

Particle Confusion and *Conjunction Confusion* occur when particle or conjunction characters are found in a word which causes the wrong Grammar Function to be chosen, and *Grammar String Invalid* occurs when a Grammar String that does not exist in the Grammar Function Dictionary is created.

The 15% error rate can almost all be accounted for as the character sets of the Japanese and Korean language been Romanized was why *Particles* and *Conjunctions* ended up being confused. In later development of the concept, Unicode was used making it impossible of characters to be confused as there is a unique binary code for every single character in the Japanese and Korean character set. This however did by no means achieve perfect translation. Something that the above table does not represent is the new kind of error that persisted to become a problem as sentences became more complex. These were logical errors, which were made because words were used incorrectly, executing some very unnatural translations in regard to word and grammar composition, in particular idioms and complex grammar use were not at all translatable. Finally, in spite of overcoming grammatical errors to only encounter many logical errors, the trial was a success. The general concept of the translator did work and further development to overcome the logical errors was possible as can be seen later.

4.2 Performance Report

4.2.1 Dexterity

The translator's dexterity is its key strength above any other aspects. The Fish Bone Composition which built upon a Spine, Bones, Muscles and even Organs in future development sources from Words and then is mapped out finally with the blueprint of the Grammar Skeleton, making a high degree of translation accuracy obtainable. Let's now go through some snippets of the algorithm to truly see the translator's remarkable dexterity.

Word Tagging

Word Tagging was indeed one of the principle methods of overcoming logical errors in translation. Treating the word as an object and tagging it with various properties can allow it to be treated properly in a sentence, where certain conventional rules may apply. Take for instance this simple example:

The particle "で" in Japanese is used to represent a) a way of doing something or b) a place something is done at. However in Korean there are two totally different particles to represent these two situations, '(으)로' or '에서'.

バスで行きました。

I went **by** bus.

버스로 갔습니다.

図書館で勉強しました。

I studied **in** the Library.

도서관**에서** 공부했습니다.

Using this predefined method from the Support Class *Rule Gate*, the previously described problem can be solved so there is no confusion between the two possible particles and a logical error can be avoided.

```
// Determines whether particle 에서 or 로 is required
public String eseoOrRo(Word word) {
      if (word.getNounType() == "Location") {
            Result = word.getKorWord() + "에서";
      } else {
            if (word.getPachimInd() == 1) {
                  Result = word.getKorWord() + "으로";
            } else {
                  Result = word.getKorWord() + "로";
            }
      }
      return Result;
}
```

Notice how the Word Class variable entered into the method is called upon its properties to make decisions. If the word is a *Location*, then '에서' is concatenated, if the word is not then '(으)로' is concatenated. Notice also that the character '으' is added depending on whether there is a *Pachim* present for the noun.

Fish Bone Composition

Here is a snippet of code for the Procedure Class *Translate*. It performs the *Bone* translation, known as case 7 for when the grammar string "V-N-は A-". The following grammar string represents a Verb Modified Noun + Particle + Adjective.

```
case 7: // "V-N-は A-"
                        Output[sentNum] =
WR[sentNum][click].getKorWord();
                        Output[sentNum] = Output[sentNum]
                              + RG.unOrNun(WR[sentNum][click+1]);
                        Output[sentNum] = Output[sentNum]
                                    +  RG.adjEnd(WR[sentNum][click+2],
                                    Integer.valueOf(mIndex.charAt(co
                                    unt)));
                        click = click + 3;
                        System.out.println(Output[sentNum]);
                        break;
```

Notice how these words are pulled from the *WR* (Word Register) in their designated order using the variable *click*. *Click* is used to identify the required words for the *Bone* being translated, while also taking into account the *Bones* that have already been translated. This prevents the use of the same words in each *Bone* and gives the translator the ability to translate multiple *Bones* for each sentence. Which of course is required if one wishes to construct a Fish Bone Composed Translation. The second important aspect to note is once *Words* are pulled from the *WR*, the Korean word can be obtained using the method *getKorWord(),* and if needs be, the Support Class *Rule Gate* also has its methods invoked inputting the Container Class *Word* as the operand to perform what ever modification is required to remain in the conventional rules of the Korean language. Each time an operation is performed on a word and a particle is added to it, it is concatenated to the final Output array and referenced to the variable *sentNum* (Sentence Number). The very last word in every *Bone* is modified according to the variable *mIndex* (muscle index

number). This alteration is done so the translated *Bone* can be attached appropriately to the *Grammar Spine* of the index.

Following the last word been modified, such methods such as *developVALink()* below are used to concatenate any further grammatical information which is required for the *Bones* particular position in the *Sentence Spine*. Here when case 2 is chosen, the modified *Adjecti*ve which is the last word in the *Bone*, now has the character '가' is also concatenated to it.

```
// Develops a Muscle link of Bone to Spine
      public String developVALink(int linkType, String linkValue)
{
            String VALink = null;
            switch (linkType) {
            case 2:
                  VALink = linkValue + "가 ";
                  break;
……
……..
……Other Muscle Link cases omitted
            }
            return VALink;
      }
```

This last snippet of code shown below is the actual function invoked to translate the sentence, which in other words is the method which composes the *Fish Bone Skeleton*. The *Bone* we have dealt with up to now is the first *Bone* that is concatenated to the *Sentence Spine* in case 1. Before the conjunction ' 때문에 ' for any verb or adjective the '기' form of that verb or adjective must be used. Since the earlier code prepared the end of the *Bone* word which happened to be an *Adjective*, which then went on to be concatenated (muscle linked) to '기', in this translate code, we simply have to concatenate *BonesOutput[0],* which contains the prepared Bone and Muscle Link to ' 때문에 '. Further *BoneOutputs* are retrieved and muscle linked to the *Sentence Spine*, altogether completing the *FinalOutput*, the *Fish Bone Composed Translation*, abbreviated as the FBCT

```
      public String[] translate(Word[][] WR, GrammarSkeleton[]
GSR) {

            for (int count = 0; count < 2; count++) {
                  BonesOutput = translateBones(WR,
GSR[count].getBoneIndices(),
                              count, GSR[count].getMuscleIndex());
                  int SpineIndex = GSR[count].getSpineIndex();
                  switch (SpineIndex) {
                  case 1: // "-から-。"
                        System.out.println("spine 1 chosen");
                        FinalOutput[count] = BonesOutput[0] +
                        " 때문에 " +      BonesOutput[1];
                        break;
……….
……
……..Other Sentence Spine cases omitted
            }
            return FinalOutput;
      }
```

4.2.2 Speed

The speed of the translator is relatively fast, as a translation can be executed in real time without the user waiting impatiently. However, true understanding of the completed model's speed is still largely unknown. This is due to many reasons, such as unknown message length (1 – 80 characters) or the translator's database not yet being fully completed, which can incur not yet known time penalties on translation. This of course can be overcome by efficient searching and referencing of data within the database, but of course in spite of all this, until the translator has been fully completed to handle the full complexity of the Japanese and Korean language, definite time values for a translation can not be given. The incomplete model in reference to this report is capable of translating an intermediate level of Korean is able to execute a translation of approximately 80 characters in length almost instantly in the blink of an eye. So there is no extreme need to worry at this point of the translator's ability to not be able to perform in real time.

4.2.3 Versatility

Versatility deals with how easily the translator is updated in consideration of how dynamic or fixed parameters of the algorithm are. At this point in time the parameters have rather fixed values, meaning several changes can also be needed within different parts of the algorithm upon making a change to one of the key parameters. It is not a serious hassle as the algorithm is still in its development stage, however if this translator was to become a commercialised service, then overhead algorithms to update the algorithms key parameters when an update is made are necessary, thus making parameters passable between classes and relieving the person updating the algorithm of any further duties. One more suggestion on this subject, further work could be done on the Procedure Class *Sentence Strip*, so it could identify, based on the text it analyzed, the minimal parameter values required when carrying out the execution of the rest of the algorithm which involves all the classes that follow. This would eliminate empty register spaces and unnecessary execution of loops which as a result consume more processor time and memory to perform the same original task.

4.3 Market Analysis

4.3.1 Survey Method

Within the host city of this research, Palmerston North New Zealand, 60 Japanese citizens took part in a ten question survey to gauge the potential market of this Translation service. A much larger scale market analysis should of course be conducted in Japan to really gain true insight into how much potential this project has, however this brief market analysis conducted here in Palmerston North is capable of demonstrating whether the project should be further continued. The ten questions delve into: if such a translator is needed, the pricing that should be associated with it, how it should be implemented, and if the market will continue to exist in the future, and so forth. The survey was conducted in Japanese eliminate possibilities of subjects misunderstanding the survey questions.

A majority of target subjects for the survey were University students from either Massey University English Learning Centre or International Pacific College in Palmerston North. This is not a good representation of the Japanese population; however the opinion of the subjects does carry a heavy weight, as they have come to New Zealand and are non native speakers of English. Due to this fact they can really give insight into whether they would find a text translator in their cellular phone necessary in their everyday life. Also one should take into consideration when interpreting this data that the subject Japanese citizens are already overseas, therefore they may feel more confident about associating with speakers of another language with little knowledge of that language. Citizens of Japan who have never experienced this situation will most probably feel even more inclined than the subjects surveyed to use the Translator put forward in this project.

4.3.2 Survey Analysis

This survey analysis takes you through each question individually to assess its importance regarding the project. Included also are graphs to help you visually see the data to draw your own conclusions. Lastly, the final results of the survey can be viewed in both English and Japanese in the Appendix if one would like to look deeper into the facts.

What are you overall thoughts on the service described?

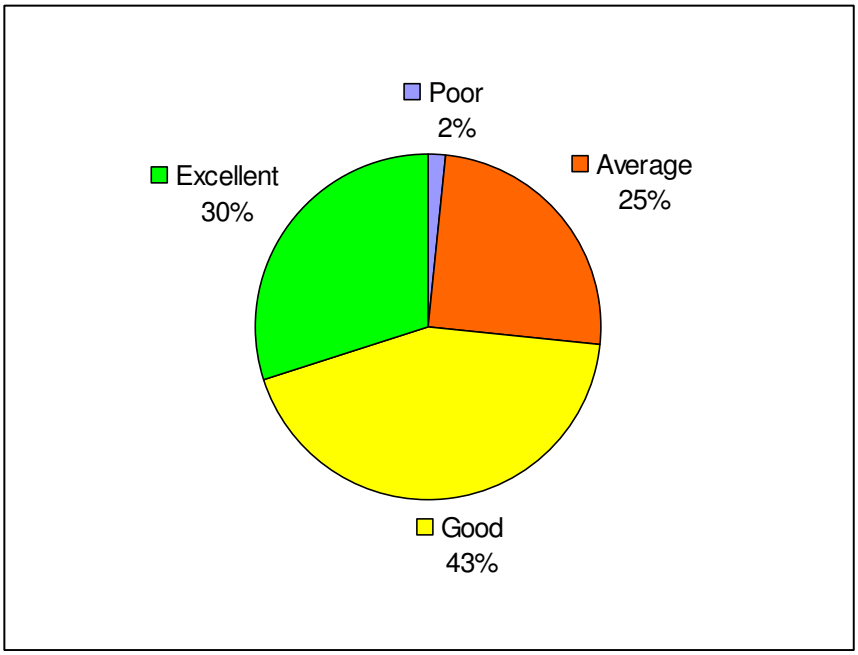

Fig 11. Pie graph representing thoughts on the service of those surveyed

From Figure 10, it can be seen that at least 73% of the surveyed subjects feel positively towards the general idea of the project, regarding it as *Excellent* or *Good*. On the other hand 27% feel the project could not benefit them to any substantial extent and have regarded the general idea of the project as either *Average* or *Poor*.

If you were travelling to Korea would you be interested in using this service?

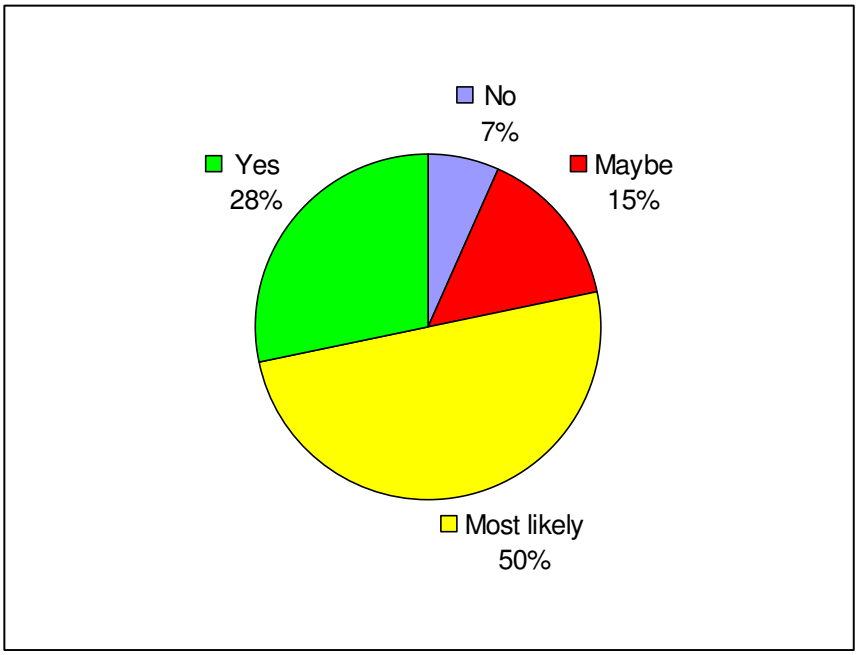

Fig 12. Pie graph representing interest in using the service of those surveyed

From Figure 11, a majority of Japanese citizens feel they could make use of the translator if they were travelling to Korea. 28% even answer that they would definitely use this service if it were provided. This is a very positive result for the project as Japanese citizens by far are the biggest market for the tourism industry of Korea. Effective marketing of the translator could really see some good returns.

What would you consider as the most important factors in choosing this product?

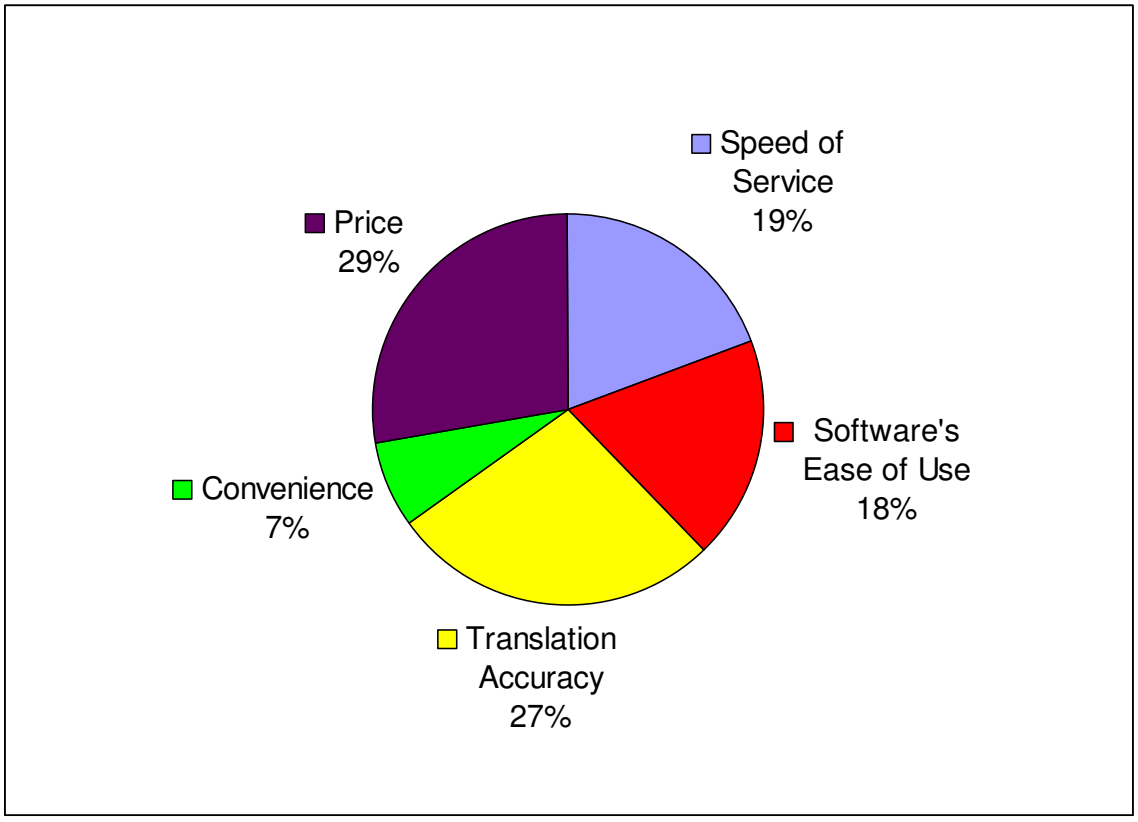

Fig 13. Pie graph representing consideration of most important factors of those surveyed

As can be seen in Figure 12, *Translation Accuracy* and *Price* were the most significant sell factors associated with the product. Following that the *Speed of Service* and the *Software's Ease of Use* trailed equally behind. General *Convenience* of having the product implemented in a cellular phone was not regarded as very important in the presence of other factors. However the survey perhaps did not stress enough that having the product incorporated into the cellular phone was a key part to the product implementation. As it is an add-on module to the Cellular phone offered at Korean airports upon entry to Korea. However this perspective is of someone who is building the device rather than using it.

What extra features would you find most useful in this product?

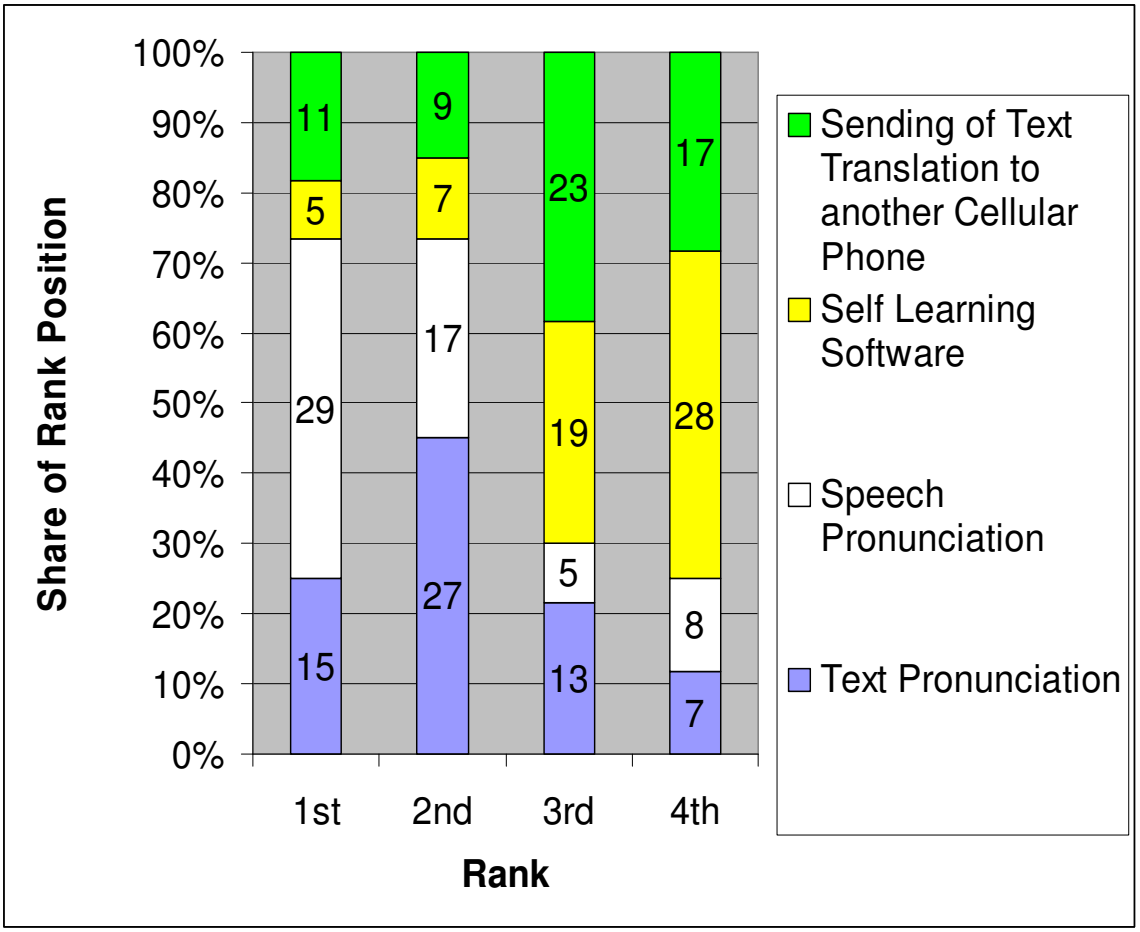

Fig 14. Representation of favoured add-on features of those surveyed

Subjects of the survey were asked to rank the 4 mentioned add-on features as how important they consider them for the translation service. Not surprisingly the *Speech Pronunciation Service* easily took first position. This is a service where a synthesized voice file is prepared with the text during translation, and the user can simply play back the Japanese to Korean translation as a sound file as they desire. Second was *Text Pronunciation*, which gives the user the chance to pronounce the translated text by using Japanese characters to display it. This of course provides more interaction between the user and the person who the user is communication with; this in turn making it a some what a more fun and educational experience, promoting the learning of the Korean language. Third was the ability to *send the translated message to another phone*, so long

distance translation communication could be achieved; for instance, if you needed to explain something to a Korean person who was to assist you from a far away location. The least favoured add on feature was the *Self Learning Software*. This is not a bad thing as *Self Learning Software* would most likely be the most difficult thing to implement, and is more or less what Sony tried to achieved with the PSP Talkman. This trend also suggests that a majority of Japanese citizens do not wish to really learn the Korean language, but wish to be able to communicate with the Korean people during their stay in Korea. Therefore any add on features to the project should be designed with this in mind.

If you were to pay for this service, would you prefer to be charged based on time used or per translation?

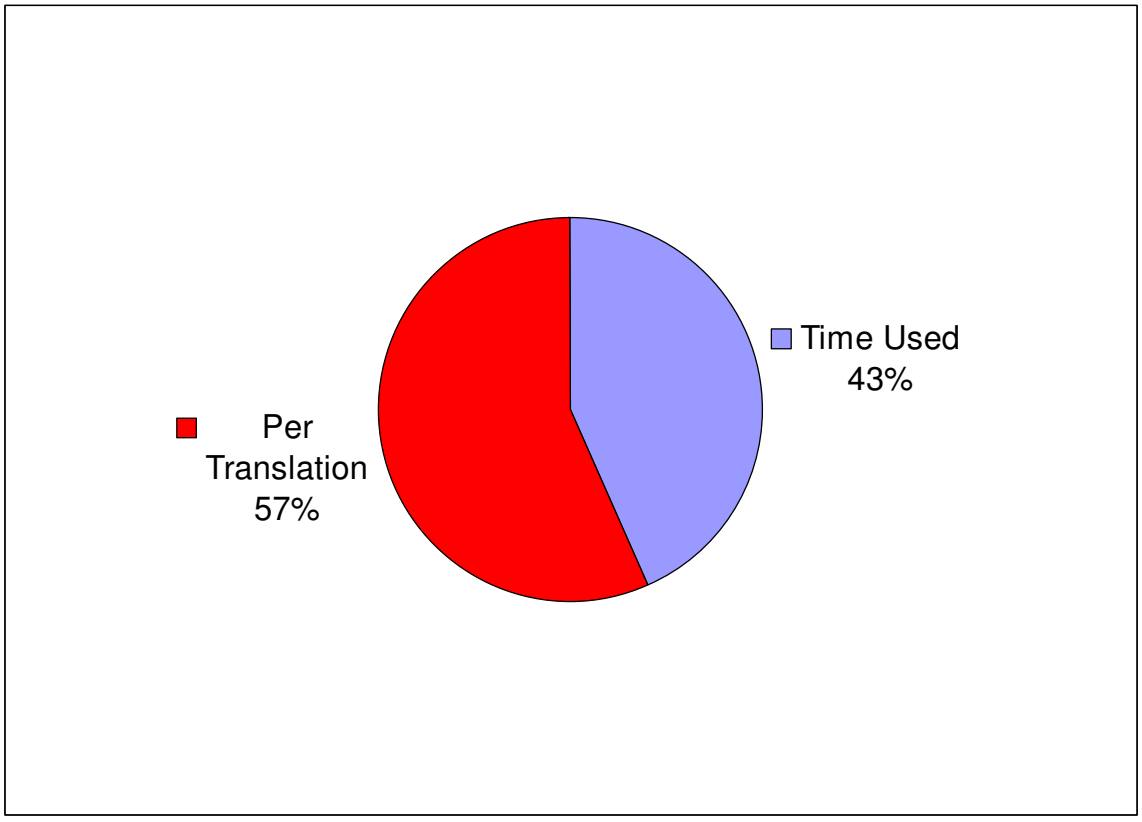

Fig 15. Pie graph representing preferred method of payment of those surveyed

Figure 14 shows most Japanese citizens would prefer to pay for the service based on how many translations they request. This suggests that they would like to have control over how much they were charged for the service. Even though they feel the need for a translator they are still not sure to what extent they will use it which is why the payment per translation option is more favourable. However both options should be available in the final implementation as they both are desired options for Japanese citizens.

If charged for time used, how much would you expect to pay for one weeks use?

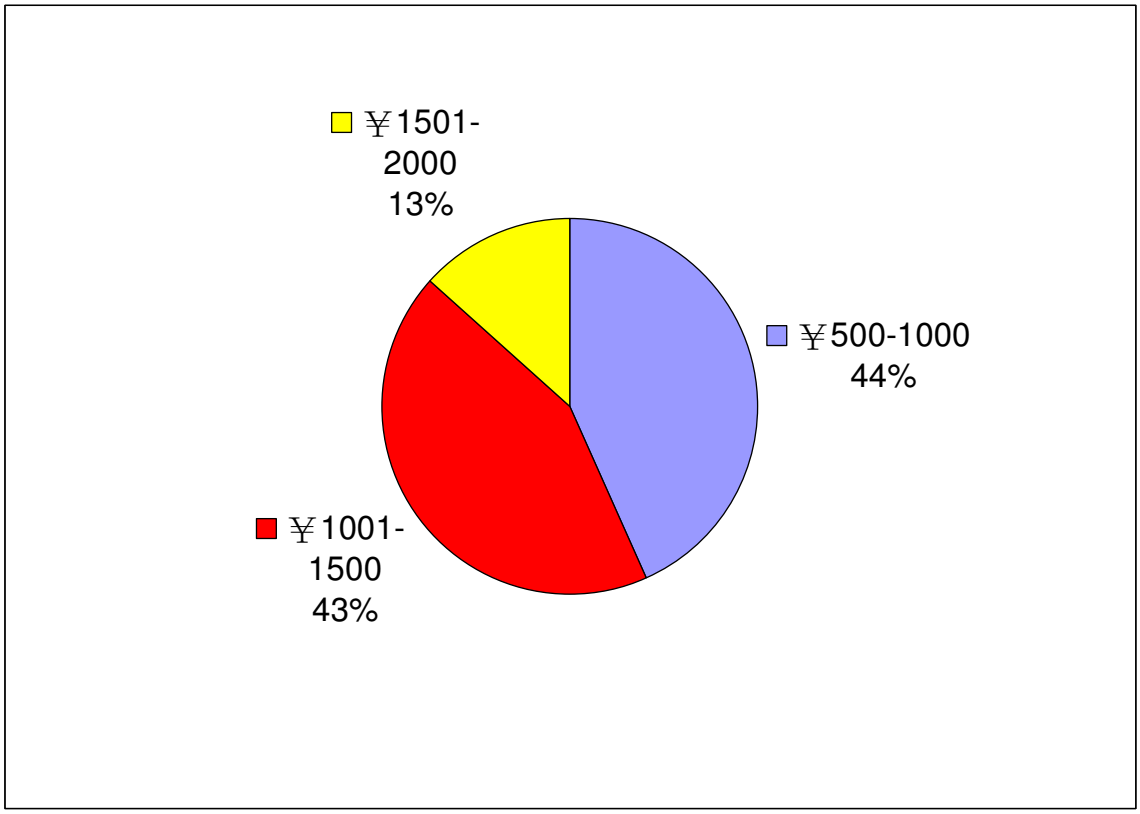

Fig 16. Pie graph representing expected payment of weekly use of those surveyed

For this question, the subjects of the survey had the choice of choosing one of the three price brackets, or listing their own expected price they would expect to be charged. As can be seen in figure 15, the price bracket of ￥500-1000 and ￥1001-1500 panned out fairly evenly. From this, it can be fair to say that Japanese citizens could expect to pay ￥1000 per week of use of this translator. Even when concerning the prices suggested by those who did not select a price bracket, their suggested prices averaged out to be about ￥1000. This further backs up that ￥1000 per week is a reasonable fee.

If charged per translation of one text message (Approximately 80 characters), how much would you expect to pay?

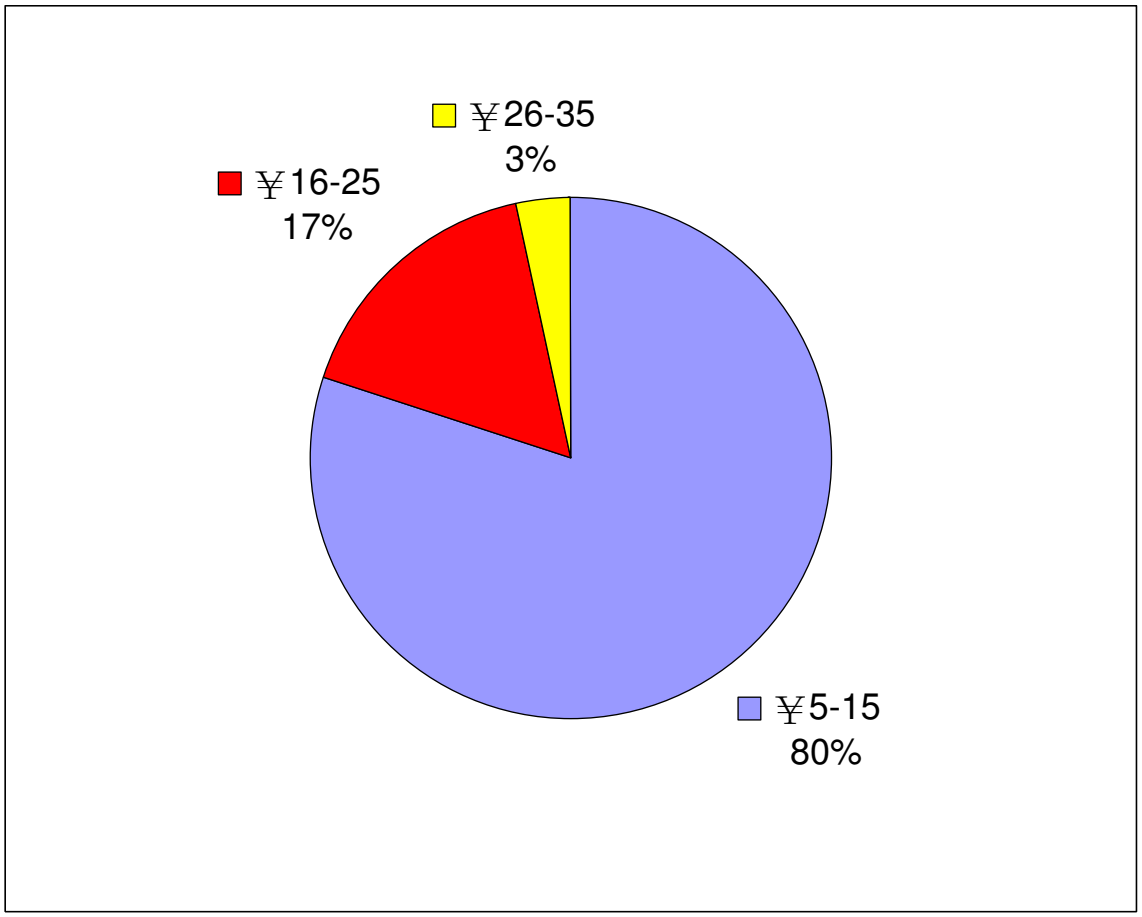

Fig 17. Pie graph representing expected payment of per translation use of those surveyed

As a counterpart to the previous question, the subjects of the survey had the choice of choosing one of the three price brackets, or listing their own expected price they would expect to be charged. A majority felt the very cheapest option would be the most they would be expected to pay for such a service. This could be due to the fact the standard pricing of a normal text message in Japan is usually only ￥1 or less. In consideration of the previous two questions, the fees implemented for such a translation system should be considered carefully to capture the both types of users; those that are unsure how much they will use the translation service (per translation charged) and those who intend to rely on it a great deal (weekly charged).

Do you intend to travel to Korea in the near future? If yes, for what purpose?

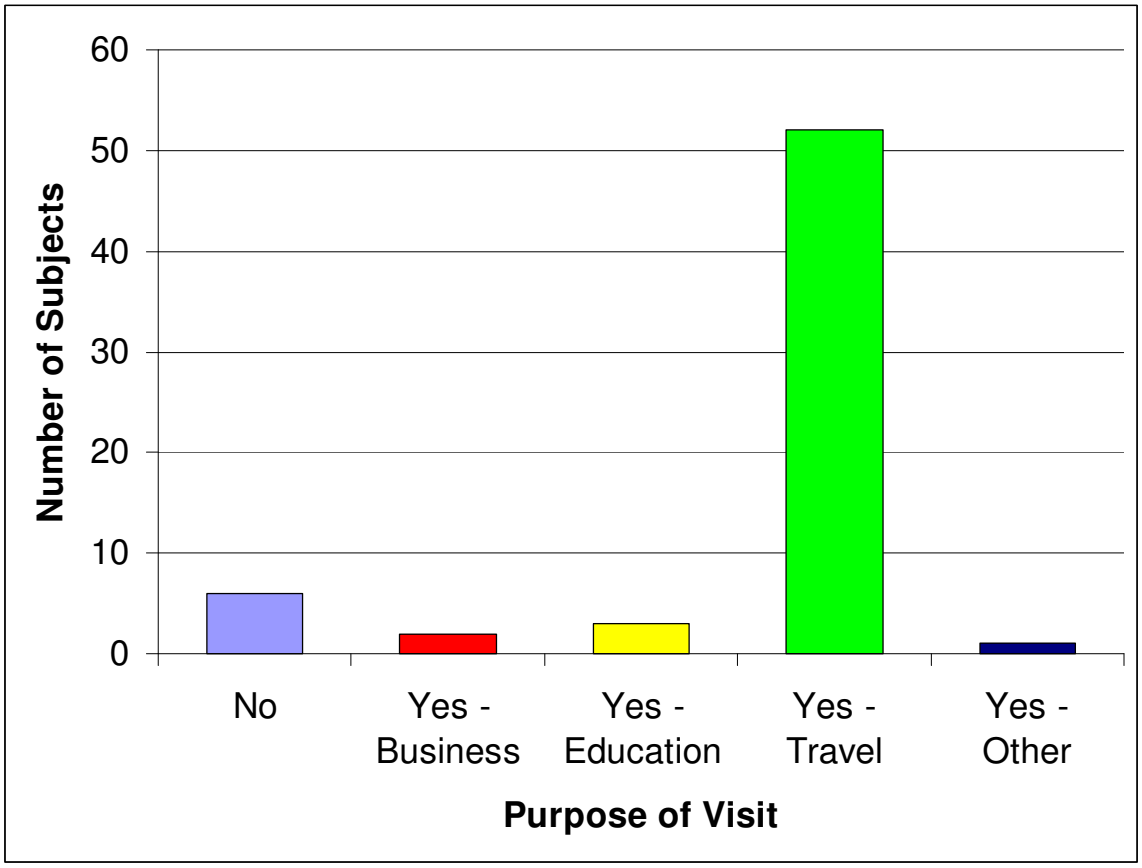

Fig 18. Bar graph representing the travel intentions of those surveyed

This question surmounted surprising but welcome results. Almost all the subjects of this survey intend to visit Korea on the basis of travel. This without doubt proves that the flow of Japanese tourists entering Korea can be expected to continue so the tourism industry of Korea should gear themselves up to with such ideas delivered by this project to take advantage of all the possible opportunities that exist. Data from the KTNO (Korean National Tourism Organization) later shown in this report also matches the results of this survey, with approximately 80 – 90% of visits to Korea by Japanese citizens in the months of February to July 2006 being for the purpose of travel. This information can be accessed by anyone who follows the KNTO website link provided in the Reference section of this report.

Have you purchased, seen or heard of a similar product?

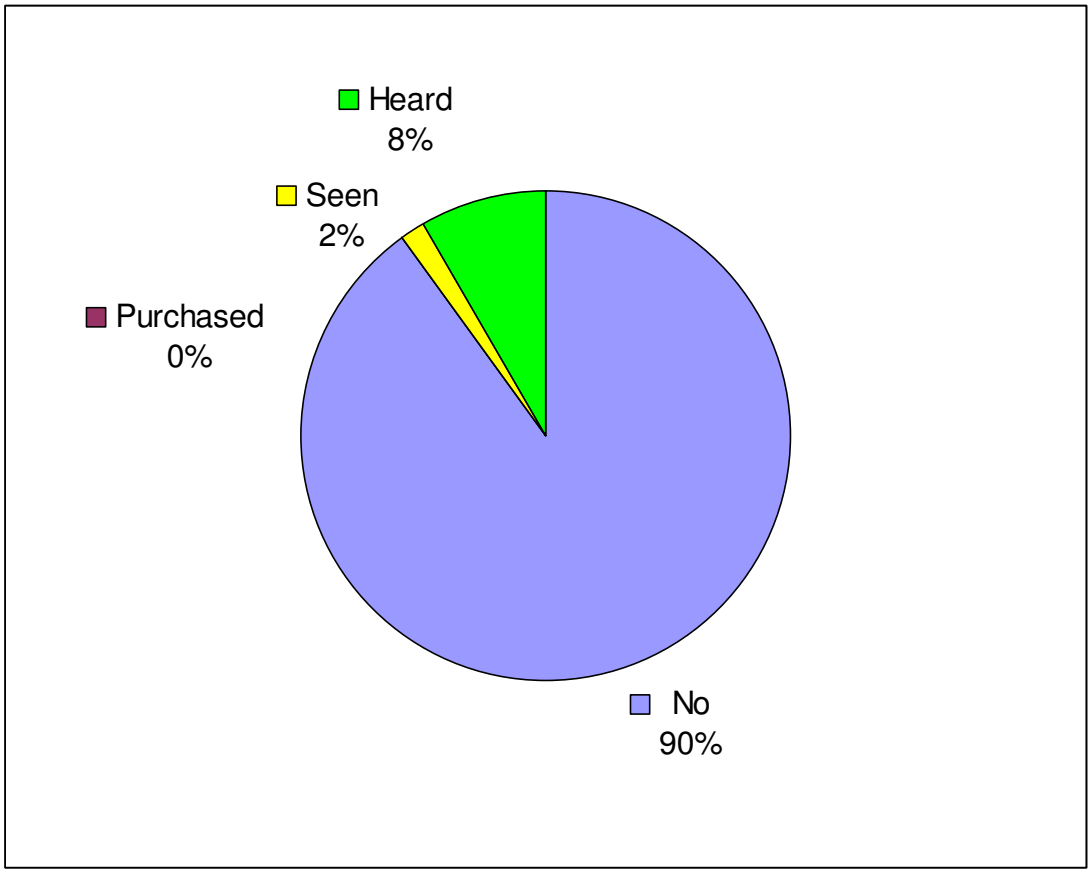

Fig 19. Pie graph representing similar services that those surveyed have encountered

Most Japanese citizens have never in their lives come across such a service available to them. Those that had heard or seen of a similar product commented that they had seen or heard of a similar product that translated Japanese into English or vice versa through the Newspaper or on Television. However no one stated that they had ever seen, heard or purchased such a product that was associated with the translation of Japanese to Korean.

Please circle your age group

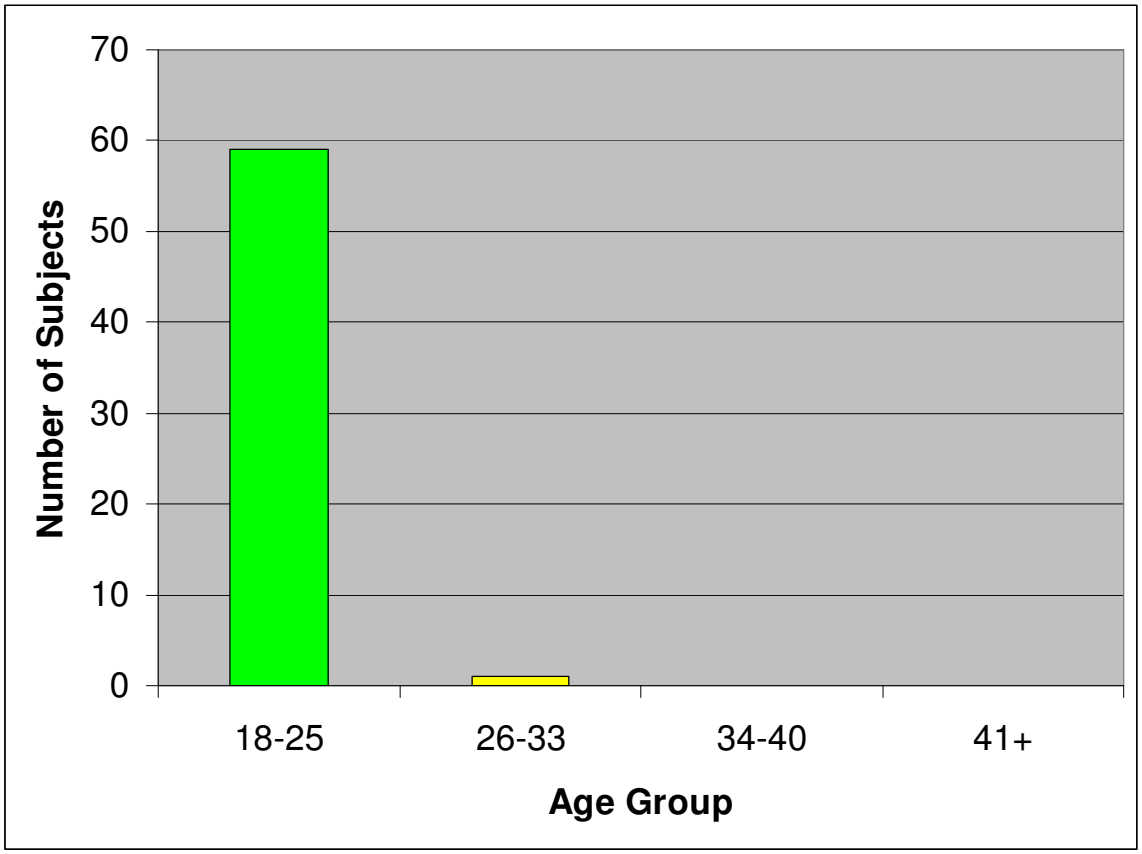

Fig 20. Bar graph representing the age bracket of those surveyed

Last of all to consider is the age group involved in this survey. As mentioned previously, most who took part were university students, which puts them in the age bracket of 18-25 years of age. This aspect should not be deemed a negative thing though, as this is the younger part of the Japanese population and a majority of them desire to travel to Korea at some point in their lives, most probably when they are in a better financial position to do so.

4.3.3 Market Projections

The following graph and table (Figure 20 & Table 2) illustrate how many arrivals to Korea in the months between February to July of 2006 were Japanese citizens. It is quite easy to see that Japan by far is Korea's biggest market for tourism. (Data obtained from the Korean National Tourism Organization [6])

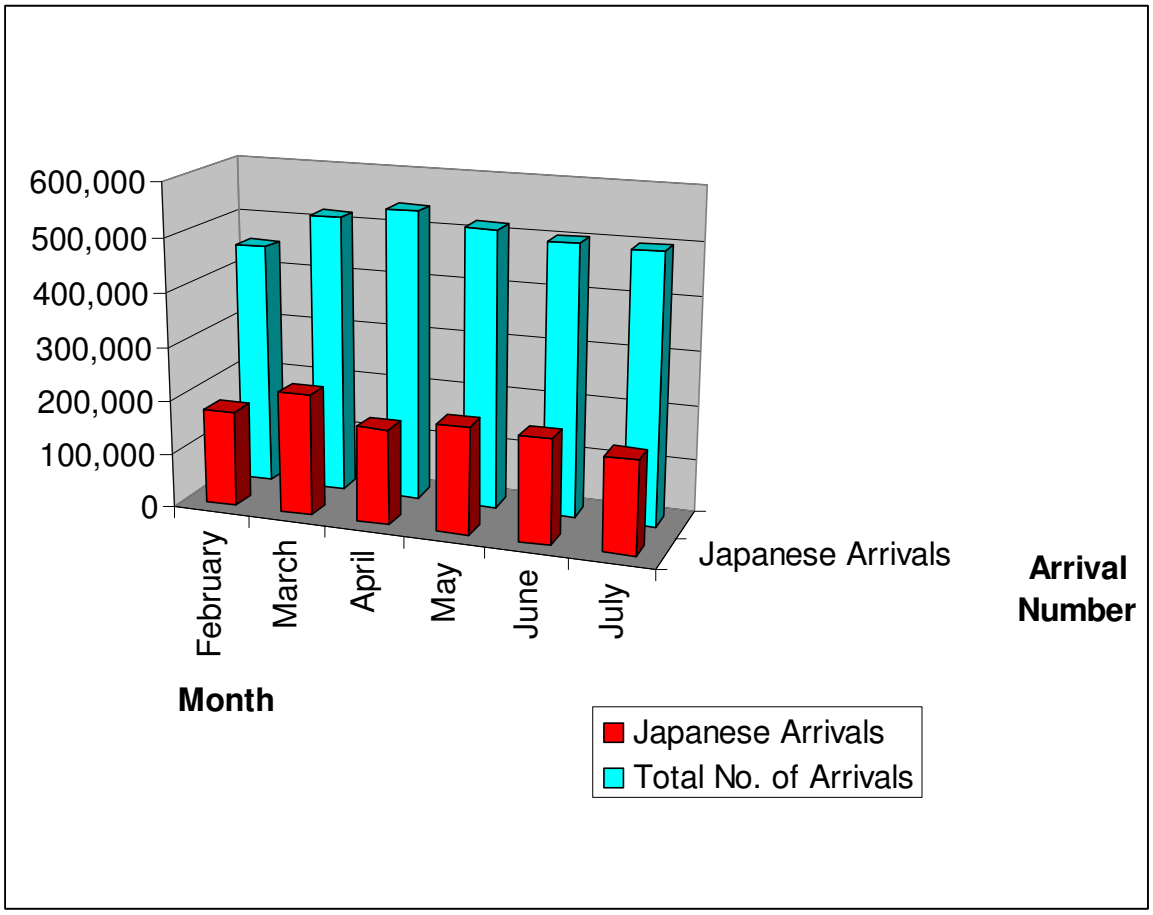

Fig 21. Bar graph representing how many of the total arrivals in Korea are Japanese citizens on a monthly basis

	Japanese Arrivals	Total No. of Arrivals
February	177,561	453,524
March	224,445	518,041
April	176,709	539,635
May	197,121	516,238
June	192,604	502,267
July	173,272	498,441
Average	190,285	504,691

Table 3. Arrival data of Japanese citizens and total number of foreigners entering Korea

We can derive some basic estimations of the project's value without regarding its set up cost to market and so forth using the data gathered from the Korean National Tourism Organization and the results of the survey conducted here in New Zealand. The average number of visitors per month was 190,285. According to the survey 28% of Japanese would be willing to use the translation service, out of the average visitors per month this is 53,280 using the translation service. If we assume they would stay for a week and were to pay the average expected price according to the survey of ¥1000 for the service on a weekly basis, then this is ¥53,280,000 (450,350 USD) per month generated. If you include those in the survey who answered *Most Likely* bringing up the market to 78% (148,422 customers per month) then this triples this figure again to ¥148,422,000 (1,254,827 USD) generated per month.

These figures are purely potential and do not take in account any variables such as market competition and if the service is effectively marketed to its true potential. Also it does not include those who would use the translation service for more than a week (which would further increase profit, for instance if customers were to use the service for two weeks instead of only one week we can double our previous calculations of potential returns). All that should be taken away from this market projection is that there is a definite market there for this type of translation service so it should be further investigated into for the benefit of the Korean Tourism industry.

5 Conclusions

The translator focuses on accuracy and portability to achieve ease of commercialization. So far the translator has achieved a high degree of dexterity for text translation with the use of *Fish Bone Composition*; nonetheless continued efforts need to be made in order to remove the common causes of errors, particularly those that are logical rather than grammatical.

A majority of errors can be avoided by further by extending the Container Classes *Grammar Skeleton* and *Word* so more information can be provided in regard to their correct usage and staying with in the conventional rules of the Korean language. However building Container Classes that store large amounts of data to increase accuracy has a serious cost on the execution time of the translation. Thus development of dynamic Container Classes should be the next step to take in this regard. A dynamic Container Class could be constructed only in respect to what is required from it later in the translation. It will require some sort of pre-emptive mechanism so the Procedure Class *Sentence Strip* could know in advance how to construct the Container Classes. One way this could be implemented is by first building the *Grammar Skeleton* class before the *Word* classes associated with it. The MLSN and the *Bones* from the *Grammar Skeleton* class could then be found and used as references to figure the data that is required for each *Word* class constructed.

Also further improvements could include updating the database to enable the translator to handle very advanced grammar and word types beyond its current comprehension. To make the algorithm run more efficiently, in respect to database searching and building of Container Classes, designing another Support Class called *Efficiency* is worth consideration. Methods that are designed in the *Efficiency* class should be public for all of the classes in the algorithm to access and run more efficiently, resulting in the translation execution requiring less processing time. Processing time is something that must be considered carefully as the translator has not yet been tested under extreme traffic conditions that would be present if it were to be commercialized. So to better

improve its chances of success in this situation, the scripting of the Support Class *Efficiency* should of course be given due consideration.

The Market Analysis showed very positive results and it really encourages further investigation into the potential market and most appropriate implementation for this translation service. For the Korean tourism industry, Japan is the single most important market holding by far the most visitors to Korea every year. In conjunction with the Korean Wave, which has continued to promote Korea as a popular tourist destination over Asia, in particular Japan, hence the environment for the development of this translator is ideal.

The tourism and IT industries of Korea are booming and competitive, this project provides a service which these two industries can converge on, rippling many direct and indirect benefits to the Korean economy at several different levels. Further polishing to the FBCT algorithm simultaneously with its implementation as a service and effective marketing to the citizens of Japan should see this project take full flight in the near future.

6 References

[1] Gianni Lazzari, Alex Waibel, C. Zong, "Worldwide Ongoing Activities On Multilingual Speech to Speech Translation", *ITC-irst Sensory Interactive Systems Division Trento Ital, CMU Interactive Lab Pittsburgh US, National Laboratory of Pattern Recognition Beijing China.*

[2] Jae-Woo Yang, Jun Park "An Experiment on Korean-to-English and Korean-to-Japanese Spoken Language Translation" *Human Interface Department, ETRI, 161 Kajong-dong, Yusung, Taejon, 305-350, Korea.*

[3] http://www.sktelecom.com/eng/cyberpr/press/1198073_3738.html
Visited 19/9/2006, article concerning the release of the SCH-V920 Real World Phone

[4] http://www.gamespot.com/psp/puzzle/talkman/review.html
Visited 10/10/2006, review by GameSpot of the Sony PSP Talkman

[5] www.wikipedia.org
Visited 13/7/2006, article concerning the Conventional Methods of Translation

[6] http://www.etourkorea.com/jsp/eng/about/research/research01_02_01.jsp
Visited 17/10/2006, The Korean National Tourism Organization home page, has data regarding Immigration Statistics of Korea.

[7] www.toshiba.com
Visited 14/10/2006, Toshiba's home page, accessed for phone information

7 Acknowledgements

7.1 People

Supervisors:

Dr. Liyanage De Silva

Mr. Amal Punchihewa

Translation Assistants:

Miss. Ryoko Hoshino

Miss. Mifa Hong

Surveyors:

Miss. Jeung Hwa Kwon

Miss. Seul Hwa Lee

Video Demonstration Actresses:

Miss. Ryoko Hoshino

Miss. Mifa Hong

7.2 Resources

www.developers.sun.com
Resource site for J2SE & J2ME development

www.mysql.org
Resource site for MySQL database development

Japanese Grammar
D. Corder, J. Short, G. Wells, C. Roughan, Heinemann Education, 2001, New Zealand
Resource for development of Translation Logic

Easy to Learn Korean, Vol. 1-6
Sungkyunkwan Language Institute (South Korea)
Resource for development of Translation Logic

8 Appendix

8.1 Survey

The survey consists of two parts. Firstly is the deployed survey which was distributed and explained in Japanese to the subjects of the survey. The subjects once answering each of the ten questions could also make a comment regarding the question. Many of the subjects did this several times throughout the survey giving further insight. Comments are not included in this report but they were helpful in drawing conclusions in the Market Analysis. Secondly there are the results which are the total tally of all surveys combined into one English version. All graphs in the Market Analysis in the report concerning the survey's ten questions were derived from this.

8.1.1 Deployed Survey (Japanese Version)

マーケティング研究のアンケート

サービス名:	携帯電話のメール機能で日本語を韓国語に訳すこと。
サービス部門:	通訳技術
施設:	**Massey University**
	Institute of Information Sciences and Technology
	Private Bag 11 222
	Palmerston North
	New Zealand
検査官:	**Steve Manion, Hannah Kwon, Seulhwa Lee**
日付:	**8/10/2006**

サービス説明:

これはマッセイ大学遠隔通信工学部の卒業論文のためのアンケートです。私は日本人観光客が韓国に訪れるにあったって、より快適な滞在が出来るように、日本語から韓国語の翻訳機能の付いた携帯電話の開発を研究しています。

この携帯電話は、日本人が韓国に訪れたときに、韓国の空港で携帯電話をレンタルするか、または事前にオンラインサービスで予約をしておくと、あなたの日本の携帯電話の電話番号がそのまま韓国でも利用でき、更にオプションとして日本語から韓国語の翻訳機能が付いてくるサービスです。

翻訳機能は、韓国語で使用したい単語や長文をメール機能を使い、８０字以内で日本語で送信すると、韓国語に翻訳されて返信されてくるというシステムです。

ID	質問	選択	コメント
1	上記で述べたサービスについてどう思いますか？	良くない まあまあ 良い 素晴らしい	
2	もし、韓国に訪れたら、このサービスを使ってみたいですか？	使いたくない あまり使いたくない 多分使いたい 使いたい	
3	もし、このサービスを使うとしたら、何が一番大切だと思いますか？ (３つ選んで下さい)	翻訳する速度 簡単に使えること 翻訳能力 便利さ 価格	
4	このサービスにオプションが必要だと思いますか？ (必要だと思う順番を１～４でランク付けして下さい。)	韓国語の読み方が送信されてくる機能 ☐ 音声翻訳機能 ☐ 自己学習ソフト ☐ 翻訳メールを送れる機能 ☐	
5	このサービスを利用するに当たって、指定された期間で支払うのと、メールの数で支払うのとどちらが良いと思いますか？	指定期間 翻訳メールの数	

ID	質問	選択	コメント
6	一週間無制限で利用可能なら、適切な値段はいくらだと思いますか？３つの中から選ぶか、それ以外の場合は、適切な値段を記入してください。	￥500-1000 ￥1000-1500 ￥1500-2000 ＝＝＝＝＝＝＝＝＝＝ ￥_____（適切な値段）	
7	翻訳メール１通なら、いくらが適切な値段だと思いますか？（各翻訳メールの長さが８０字位です。）３つの中から選ぶか、それ以外の場合は、適切な値段を記入してください。	￥5-15 ￥16-25 ￥26-35 ＝＝＝＝＝＝＝＝＝＝ ￥_____（適切な値段）	
8	将来韓国に行きたいと思っていますか？その目的はなんですか？	いいえ ＝＝＝＝＝＝＝＝＝＝ はい – ビジネス - 教育 - 旅行 - その他	
9	このようなサービスを買ったり、見たり、聞いたりした事がありますか？複数選択可能です。	いいえ 買ったことがある 見たことがある 聞いたことがある	
10	年齢層を○で囲んでください。	18-25　26-33 34-40　41+	

8.1.2 Results (English Version)

ID	Question	Selection	Results
1	What are your overall thoughts on the service described?	Poor Average Good Excellent	1 15 26 18
2	If you were travelling to Korea would you be interested in using this service?	No Maybe Most likely Yes	4 9 30 17
3	What would you consider as the most important factors in choosing this product? (Please select three)	Speed of Service Ease of Use Translation Accuracy Convenience Price	35 33 49 13 50
4	What extra features would you find most useful on this product? (Rank in order of choice, 1 = most useful)	Text Pronunciation ☐ Speech Pronunciation ☐ Self Learning Software ☐ Sending of Text Translation to another Cellular Phone ☐	2nd 1st 4th 3rd
5	If you were to pay for this service, would you prefer to be charged based on the time used or per translation?	Time used Per Translation	26 34

ID	Question	Selection	Response/Comments
6	If charged for time used, how much would you expect to pay for one weeks use?	¥500-1000 ¥1000-1500 ¥1500-2000 ¥_____	23 23 7
7	If charged per translation of one text message (Approximately 80 characters), how much would you expect to pay?	¥5-15 ¥16-25 ¥26-35 ¥_____	48 10 2
8	Do you intend to travel to Korea in the near future? If yes, for what purpose?	No Yes - Business - Education - Travel - Other	6 2 3 52 1
9	Have you purchased, seen, or heard of a similar product?	No Purchased Seen Heard	54 0 1 5
10	Please circle your age group	18-25 26-33 34-40 41+	59 1 0 0

8.2 FBCT Algorithm (Stand Alone Version)

The algorithm denoted here is the stand alone version scripted in J2SE, which executes translations locally; by simply placing text files into the algorithm to receive a generated text file of the translation in return. The algorithm used in the actual communication protocol is of course slightly different and can be compiled into an independent deployable servlet where as this version if compiled into a servlet, relies on local input arguments as variables and so forth. Also appropriate commenting and simplification of inefficiencies with in the algorithm may be lacking, particular in classes like *Sentence Strip*. The algorithm was still in the process of development at the time of this reports due publication. Future publications regarding this project of course will contain a more polished documentation of the FBCT algorithm.

The Support Class *Dictionary* is not listed as it is very large and repetitious in structure, further more with the classes that are denoted here only specific segments are revealed and where methods concerning grammar and word manipulation are concerned, only few are shown to provide examples to the reader. Due to the time and work that has gone into developing the current library of grammar and word manipulation methods, it is considered as Intellectual Property. The purpose of revealing segments of the FBCT algorithm in this appendix is only to provide readers with sufficient resources to help them understand the internal workings of the software, but not to assist them in any self-distribution or use of it through copying its source.

8.2.1 Control Class

```
import java.io.*;

public class JKT {

	public static void main(String[] args) {

		try {
			//Generate required Classes
			Dictionary JKD = new Dictionary();
			RuleGate RG = new RuleGate();
			SentenceStrip SS = new SentenceStrip(JKD);
			JKTranslate JKT = new JKTranslate(RG);

			//Gather source text information
			SS.Strip(“input.txt”);
			GrammarSkeleton[] GSR = SS.getGramSkelReg();
			Word[][] WR = SS.getWordReg();

			//Execute & Write the FBCT to output.txt
			String[] Translation = JKT.translate(WR, GSR);
			DataOutputStream outStream = new DataOutputStream(
					new FileOutputStream("output.txt"));
			for (int count = 0; count < 2; count++) {
				if (Translation[count] != null) {
					outStream.writeUTF(Translation[count]);
					System.out.println(Translation[count]);
				}
			}
			outStream.close();
		} catch (Exception e) {
			System.out.println(e);
		}

	}
}
```

8.2.2 Sentence Strip Class (Procedure)

```
import java.io.*;

public class SentenceStrip {
	private Word[][] WR;

	private GrammarSkeleton[] GR;

	public Dictionary DictLink;

	public SentenceStrip(Dictionary dict) {
		WR = new Word[10][10];
		GR = new GrammarSkeleton[10];
		DictLink = dict;
	}

	// Get Word Register
	public Word[][] getWordReg() {
		return WR;
	}

	// Get Grammar Skeleton Register
	public GrammarSkeleton[] getGramSkelReg() {
		return GR;
	}

	// Generate a word
	public Word generateWord(String jWord) {
		Word finalWord = null;
		for (int count = 0; count < 10; count++) {

			if ((jWord.equals(DictLink.JNDict[count][0]))
					| (jWord.equals(DictLink.JNDict[count][1]))) {
				System.out.println("N match!");
				Word builtWord = new Word(jWord, DictLink.KNDict[count][0],
```

```
                        Integer.valueOf(DictLink.KNDict[count][1]), DictLink.KNDict[count][2]);
                builtWord.setWordType("N");
                System.out.println(builtWord.getKorWord());
                count = 10;
                finalWord = builtWord;
            } else if ((jWord.equals(DictLink.JVADict[count][0]))
                        | (jWord.equals(DictLink.JVADict[count][1]))) {
                System.out.println("VA match!");
                int DI = Integer.valueOf(DictLink.JVAData[count][2]);
                Word builtWord = new Word(jWord, DictLink.KVADict[DI][0],
                        Integer.valueOf(DictLink.KVADict[DI][3]),
                        DictLink.JVAData[count][0], DictLink.JVAData[count][1],
                        DictLink.KVADict[DI][0], DictLink.KVADict[DI][2],
                        DictLink.KVADict[DI][1], DictLink.KVADict[DI][4],
                        DictLink.KVADict[DI][5]);
                System.out.println(builtWord.getKorWord());
                count = 10;
                finalWord = builtWord;
            }
        }
        return finalWord;
}

// Generate a Grammar Skeleton
public GrammarSkeleton generateGramSkel(String spine, String[] bones) {
        GrammarSkeleton GS = new GrammarSkeleton(spine, bones);
        GS.setSpineIndex((findSpineGI(spine) - 1));
        System.out.println(findSpineGI(spine));
        GS.setMuscleIndex(DictLink.MDict[(findSpineGI(spine) - 1)]);
        for (int count = 0; count < 10; count++) {
            if (bones[count] != null) {
                GS.setBoneIndex(count, findBoneGI(bones[count]));
            }
        }
        return GS;
}

public int findSpineGI(String GS) {
```

```
	int BoneGI = 0;
	for (int count = 0; count < 8; count++) {
		if (GS.equals(DictLink.SDict[count])) {
			BoneGI = count + 1;
			count = 7;
		}
	}
	return BoneGI;
}

public int findBoneGI(String GS) {
	int BoneGI = 0;
	for (int count = 0; count < 8; count++) {
		if (GS.equals(DictLink.BDict[count])) {
			BoneGI = count + 1;
			count = 7;
		}
	}
	return BoneGI;
}

public void Strip(String inputFile) {
	try {
		StringBuffer KeyBuffer = new StringBuffer(); // Primary Buffer to catch the sentence
		StringBuffer SGSBuffer = new StringBuffer(); // Spine Grammar String Buffer
		StringBuffer BGSBuffer = new StringBuffer(); // Bone Grammar String Buffer
		StringBuffer CWBuffer = new StringBuffer(); // Check Word Buffer
		StringBuffer WBBuffer = new StringBuffer(); // Word Build Buffer
		StringBuffer CPBuffer = new StringBuffer(); // Check Particle Buffer
		StringBuffer PBBuffer = new StringBuffer(); // Particle Build Buffer
		StringBuffer RBuffer = new StringBuffer(); // Stores word possibly
		// mistaken for a particle

		InputStream is = null;
		InputStreamReader isr1 = null;
		is = SentenceStrip.class.getResourceAsStream(inputFile);
		if (is == null)
			throw new Exception("File Does Not Exist");
```

```
isr1 = new InputStreamReader(is, "UTF8");

int ch;
int chp = 0;
int wdp = 0;
int intlen = 0;
int len1 = 0;
int len2 = 0;
int len3 = 0;
int len4 = 0;
int partSize = 0;
int sentNum = 0;
int boneCount = 0;
boolean Wok = true;
boolean Pok = false;
boolean recoverChar = false;
boolean secondWord = false;
boolean recoverPart = false;
String WordTag = null;
String PExistanceCheck = null;
String ResWord = null;
String[] BoneCollector = null;

while ((ch = isr1.read()) > -1) {
      KeyBuffer.append((char) ch);
      if ((chp != 0) & (Wok == true) & (Pok == false)
                  & (secondWord == false)) {
            CPBuffer = new StringBuffer();
            PBBuffer = new StringBuffer();
            CWBuffer.append((char) ch);
            partSize = 0;
            len2 = 0;
            len1++;
            System.out.println("character appended to CWBuffer");
            String checkWord = CWBuffer.toString();
            for (int count = 0; count < 10; count++) {
                  if ((checkWord.regionMatches(0,
```

```
                        DictLink.JNDict[count][0], 0, len1))
                        | (checkWord.regionMatches(0,
                                    DictLink.JNDict[count][1], 0, len1))) {
                Wok = true;
                Pok = false;
                System.out.println("match found");
                WBBuffer.append((char) ch);
                WordTag = "N-";
                count = 10;
        } else if ((checkWord.regionMatches(0,
                        DictLink.JVADict[count][0], 0, len1))
                        | (checkWord.regionMatches(0,
                                    DictLink.JVADict[count][1], 0, len1))) {
                Wok = true;
                Pok = false;
                System.out.println("match found");
                WBBuffer.append((char) ch);
                WordTag = DictLink.JVAData[count][0];
                count = 10;
        } else {
                Wok = false;
                Pok = true;
                System.out.println("match not found");
        }
}
if ((chp != 0) & (Pok == true) & (Wok == false)) {
        String word = WBBuffer.toString();
        WR[sentNum][wdp] = generateWord(word);
        System.out.println(word);
        word = null;
        SGSBuffer.append(WordTag);
        BGSBuffer.append(WordTag);
        System.out
                        .println("Tag appended to SGSBuffer and BGSBuffer");
        WordTag = null;
        CWBuffer = new StringBuffer();
        WBBuffer = new StringBuffer();
        wdp++;
```

```
                len1 = 0;
        }
}

if ((chp != 0) & (Pok == true) & (Wok == false)) { // &((chCatch

        CPBuffer.append((char) ch);

        len2++;
        System.out.println("character appended to CPBuffer");
        String checkParticle = CPBuffer.toString();
        System.out.println(checkParticle);
        {
              for (int count = 0; count < 10; count++) {
                    if ((checkParticle.regionMatches(0,
                                DictLink.CDict[count][0], 0, len2))
                                | (checkParticle.regionMatches(0,
                                            DictLink.CDict[count][1], 0, len2))) {
                          Wok = false;
                          Pok = true;
                          System.out.println("p match found");
                          PBBuffer.append((char) ch);
                          WBBuffer.append((char) ch);
                          RBuffer.append((char) ch);
                          partSize++;
                          recoverChar = false;
                          // if (partSize < 2){

                          // }
                          count = 10;
                    } else {
                          Wok = true;
                          Pok = false;
                          recoverChar = true;
                          System.out.println("p match not found");
                    }
              }
              if (Pok == false) {
```

```
                        RBuffer.append((char) ch);

                    }
                }
                if (recoverChar) {

                    boolean isParticle = false;
                    PExistanceCheck = PBBuffer.toString();
                    ResWord = RBuffer.toString();
                    System.out.println("it is" + PExistanceCheck);
                    for (int count1 = 0; count1 < 10; count1++) {
                        for (int count2 = 0; count2 < 2; count2++) {
                            if (DictLink.CDict[count1][count2]
                                    .equals(PExistanceCheck)) {
                                isParticle = true;
                                if (DictLink.CDict[count1][2].equals("B")) {
                                    BGSBuffer

.append(DictLink.CDict[count1][count2]);
                                } else {
                                    BoneCollector[boneCount] = BGSBuffer
                                            .toString();
                                    BGSBuffer = new StringBuffer();
                                    boneCount = boneCount + 1;
                                }
                            }
                        }
                    }
                    if (isParticle) {
                        recoverChar = true;
                        System.out.println("Particle is legitimate  >>>>>");
                        RBuffer = new StringBuffer();
                        WBBuffer = new StringBuffer();
                    } else {
                        secondWord = true;
                        recoverChar = false;
                        len3 = ResWord.length();
                        intlen = len3;
```

```
                                    System.out.println("length is" + len3);
                                    System.out
                                                .println("Particle confusion, search for second
word!");

                              }
                        }
                  }

                  if (secondWord) {

                        if (intlen != len3) {
                              RBuffer.append((char) ch);
                              ResWord = RBuffer.toString();
                        }
                        System.out.println(ResWord);
                        for (int count = 0; count < 10; count++) {
                              if ((ResWord.regionMatches(0,
                                          DictLink.JNDict[count][0], 0, len3))
                                          | (ResWord.regionMatches(0,
                                                      DictLink.JNDict[count][1], 0, len3))) {
                                    Wok = true;
                                    Pok = false;
                                    secondWord = true;
                                    System.out.println("second word match found N");
                                    WBBuffer.append((char) ch);
                                    WordTag = "N-";
                                    count = 10;
                              } else if ((ResWord.regionMatches(0,
                                          DictLink.JVADict[count][0], 0, len3))
                                          | (ResWord.regionMatches(0,
                                                      DictLink.JVADict[count][1], 0, len3))) {
                                    Wok = true;
                                    Pok = false;
                                    secondWord = true;
                                    System.out.println("second word match found V/A");
                                    WBBuffer.append((char) ch);
                                    WordTag = DictLink.JVAData[count][0];
```

```
                        count = 10;
                  } else {
                        Wok = false;
                        Pok = true;
                        secondWord = false;
                        System.out.println("second word match not found");
                  }
            }
            if ((chp != 0) & (Pok == true) & (Wok == false)
                        & (secondWord == false)) {
                  String word = WBBuffer.toString();
                  WR[sentNum][wdp] = generateWord(word);
                  System.out.println(word);
                  SGSBuffer.append(WordTag);
                  BGSBuffer.append(WordTag);
                  System.out
                              .println("Tag appended to GSBuffer and BGSBuffer");
                  word = null;
                  CWBuffer = new StringBuffer();
                  WBBuffer = new StringBuffer();
                  RBuffer = new StringBuffer();
                  CPBuffer = new StringBuffer();
                  wdp++;
                  intlen = 0;
                  partSize = 0;
                  System.out.print("Second word inside array");
                  recoverPart = true;
            }
            len3++;
      }

      if (recoverPart) {
            {
                  len4++;
                  CPBuffer.append((char) ch);
                  String checkParticle = CPBuffer.toString();
                  for (int count = 0; count < 10; count++) {
                        if ((checkParticle.regionMatches(0,
```

```
                                    DictLink.CDict[count][0], 0, len4))
                                    | (checkParticle.regionMatches(0,
                                                DictLink.CDict[count][1], 0, len4))) {
                              Wok = false;
                              Pok = true;
                              System.out.println("p match found");
                              PBBuffer.append((char) ch);
                              partSize++;
                              count = 10;
                        } else {
                              Wok = true;
                              Pok = false;
                              System.out.println("p match not found");
                        }
                  }
            }
            recoverPart = false;
            len2 = 1;
      }
      if (recoverChar) {
            len3 = 0;
            String particle = PBBuffer.toString();
            SGSBuffer.append(particle);
            BGSBuffer.append(particle);
            System.out.println("character appended to GSBuffer");
            // WR[sentNum][0] = SGSBuffer.toString();
            if (particle.equals("。")) {
                  int endPoint = BGSBuffer.indexOf("。");
                  BGSBuffer.deleteCharAt(endPoint);
                  BoneCollector[boneCount] = BGSBuffer.toString();
                  GR[sentNum] = generateGramSkel(SGSBuffer.toString(),
                              BoneCollector);
                  sentNum++;
                  System.out.println("++ Sentence Complete ++");
                  CWBuffer = new StringBuffer();
                  WBBuffer = new StringBuffer();
                  CPBuffer = new StringBuffer();
```

```
            PBBuffer = new StringBuffer();
            SGSBuffer = new StringBuffer();
            BGSBuffer = new StringBuffer();
            RBuffer = new StringBuffer();
      }
      len1++;
      System.out.println("dumped character appended to CWBuffer");
      CWBuffer.append((char) ch);
      String checkWord = CWBuffer.toString();
      System.out.println(checkWord);
      for (int count = 0; count < 10; count++) {
            if ((checkWord.regionMatches(0,
                        DictLink.JNDict[count][0], 0, len1))
                        | (checkWord.regionMatches(0,
                                    DictLink.JNDict[count][1], 0, len1))) {
                  Wok = true;
                  Pok = false;
                  System.out.println("match found");
                  WBBuffer.append((char) ch);
                  WordTag = "N-";
                  count = 10;
            } else if ((checkWord.regionMatches(0,
                        DictLink.JVADict[count][0], 0, len1))
                        | (checkWord.regionMatches(0,
                                    DictLink.JVADict[count][1], 0, len1))) {
                  Wok = true;
                  Pok = false;
                  System.out.println("match found");
                  WBBuffer.append((char) ch);
                  WordTag = DictLink.JVAData[count][0];
                  count = 10;
            } else {
                  Wok = false;
                  Pok = true;
                  System.out.println("match not found");
            }
      }
      recoverChar = false;
```

```
                        }

                        chp++;
                    }
                    String particle = PBBuffer.toString();
                    SGSBuffer.append(particle);
                    System.out.println("last character appended to GSBuffer");
                    System.out.println("++ Sentence Complete ++");

                    if (isr1 != null)
                        isr1.close();
                    String line = KeyBuffer.toString();
                    System.out.println(line);

                    DataOutputStream outStream = new DataOutputStream(
                                new FileOutputStream("SSoutput.doc"));
                    outStream.writeUTF(line);

                    for (int count1 = 0; count1 <= sentNum; count1++) {
                        for (int count2 = 0; count2 < wdp; count2++) {
                            String outputArt = WR[count1][count2].getJapWord();
                            if (outputArt != null) {
                                System.out.println(WR[count1][count2].getJapWord());
                                outStream.writeUTF(WR[count1][count2].getJapWord());
                            }
                        }
                    }
                    outStream.close();
                } catch (Exception e) {
                    System.out.println(e);
                }

        }

}
```

8.2.3 Word Class (Container)

```
public class Word {

	String japWord;

	String korWord;

	String wordType;

	int pachimInd;

	String nounType;

	String wordTense;

	String verbAdjStem;

	String verbAdjPast;

	String verbAdjVol;

	String verbAdjBform;

	String verbAdjNform;

…….
……..
……... further fields omitted

	// Constructor for a Noun
	public Word(String jWord, String kWord, int pInd, String nType) {
		japWord = jWord;
		korWord = kWord;
		pachimInd = pInd;
		nounType = nType;

		wordType = null;
```

```
            wordTense = null;
            verbAdjStem = null;
            verbAdjPast = null;
            verbAdjVol = null;
            verbAdjBform = null;
            verbAdjNform = null;

…….
……..
……... further variables omitted

      }

      // Constructor for a Verb/Adjective
      public Word(String jWord, String kWord, int pInd, String wType,
                  String wTense, String stem, String past, String vol, String bform,
                  String nform) {
            japWord = jWord;
            korWord = kWord;
            pachimInd = pInd;
            wordType = wType;

            wordTense = wTense;
            verbAdjStem = stem;
            verbAdjPast = past;
            verbAdjVol = vol;
            verbAdjBform = bform;
            verbAdjNform = nform;

…….
……..
……... further variables omitted

      }

      // Returns the Japanese word
      public String getJapWord() {
            return japWord;
```

```
}

// Returns the Korean word
public String getKorWord() {
	return korWord;
}

// Returns the word type
public String getWordType() {
	return wordType;
}

// Returns the word tense
public String getWordTense() {
	return wordTense;
}

// Indicates if there is pachim or not
public int getPachimInd() {
	return pachimInd;
}

// Returns Noun Type
public String getNounType() {
	return nounType;
}

// Returns the Verb/Adjective stem
public String getVAStem() {
	return verbAdjStem;
}

// Returns the Verb/Adjective past form
public String getVAPast() {
	return verbAdjPast;
}
```

```
	// Returns the Verb/Adjective volitional form
	public String getVAVol() {
		return verbAdjVol;
	}

	// Returns the Verb/Adjective ㅂ-form
	public String getVABform() {
		return verbAdjBform;
	}

	// Returns the Verb/Adjective ㄴ-form
	public String getVANform() {
		return verbAdjNform;
	}

	// Sets the word type
	public void setWordType(String wType) {
		wordType = wType;
	}
…….
……..
……... further methods omitted


}
```

8.2.4 Grammar Skeleton (Container)

```
public class GrammarSkeleton {

	String Spine;

	int SpineGI;

	String[] Bones;

	int[] BonesGI;

	String MuscleGI;

	// Constructor of Grammar Skeleton
	public GrammarSkeleton(String spine, String[] bones) {
		Spine = spine;
		Bones = bones;
	}

	// Set the Spine Grammar Index
	public void setSpineIndex(int sIndex) {
		SpineGI = sIndex;
	}

	// Get the Spine Grammar Index
	public int getSpineIndex() {
		return SpineGI;
	}

	// Set a Bone Grammar Index according to its order in the sentence
	public void setBoneIndex(int boneNumber, int bIndex) {
		BonesGI[boneNumber] = bIndex;
	}

	// Get a Bone Grammar Index according to its order in the sentence
	public int getBoneIndex(int boneNumber) {
		return BonesGI[boneNumber];
```

```
	}

	// Get Bone Grammar Indices
	public int[] getBoneIndices() {
		return BonesGI;
	}

	// Set Muscle Index
	public void setMuscleIndex(String mIndex) {
		MuscleGI = mIndex;
	}

	// Get Muscle Index
	public String getMuscleIndex() {
		return MuscleGI;
	}

	// Returns the Spine
	public String getSpine() {
		return Spine;
	}

	// Returns a Bone according to its order in the sentence
	public String getBone(int boneNumber) {
		return Bones[boneNumber];
	}
}
```

8.2.5 Rule Gate (Support)

```
public class RuleGate {
	// Feilds
	private String Result; // Result of passing through a Rule Gate

	// Constructor
	public RuleGate() {
		Result = null;
	}

	// Determines whether particle 은 or 는 is required
	public String unOrNun(Word word) {
		if (word.getPachimInd() == 0) {
			Result = word.getKorWord() + "는 ";
		} else {
			Result = word.getKorWord() + "은 ";
		}
		return Result;
	}

	// Determines whether particle 이 or 가 is required
	public String iOrGa(Word word) {
		if (word.getPachimInd() == 0) {
			Result = word.getKorWord() + "가 ";
		} else {
			Result = word.getKorWord() + "이 ";
		}
		return Result;
	}

	// Determines whether particle 에서 or 로 is required
	public String eseoOrRo(Word word) {
		if (word.getNounType() == "Location") {
			Result = word.getKorWord() + "에서";
		} else {
			if (word.getPachimInd() == 1) {
```

```
                Result = word.getKorWord() + "으로";
            } else {
                Result = word.getKorWord() + "로";
            }
        }
        return Result;
}

// Forms Verb ending
public String verbEnd(Word word, int link) {
        if (link == 1) {
            if (word.getWordTense() == "PF") {
                if (word.getPachimInd() == 1) {
                    Result = word.getVAStem() + "습니다.";
                } else {
                    Result = word.getVABform() + "니다.";
                }
            } else if (word.getWordTense() == "PP") {
                if (word.getPachimInd() == 1) {
                    Result = word.getVAStem() + "었습니다.";
                } else {
                    Result = word.getVAPast() + "습니다.";
                }
            } else if (word.getWordTense() == "CF") {
                Result = word.getVAVol() + ".";
            } else if (word.getWordTense() == "CP") {
                Result = word.getVAPast() + "어.";
            }
        } else {
            if ((word.getWordTense() == "PF") | (word.getWordTense() == "CF")) {
                Result = developVALink(link, word.getVAStem());
            } else {
                Result = developVALink(link, word.getVAPast());
            }

        }
        return Result;
```

```
}

// Forms an Adjective ending
public String adjEnd(Word word, int link) {
	if (link == 1) {
		if (word.getWordTense() == "PF") {
			if (word.getPachimInd() == 1) {
				Result = word.getVAStem() + "입니다.";
			} else {
				Result = word.getVABform() + "니다.";
			}
		} else if (word.getWordTense() == "PP") {
			if (word.getPachimInd() == 1) {
				Result = word.getVAStem() + "었습니다.";
			} else {
				Result = word.getVAPast() + "습니다.";
			}
		} else if (word.getWordTense() == "CF") {
			Result = word.getVAVol() + ".";
		} else if (word.getWordTense() == "CP") {
			Result = word.getVAPast() + "어.";
		}
	} else {
		if ((word.getWordTense() == "PF") | (word.getWordTense() == "CF")) {
			Result = developVALink(link, word.getVAStem());
		} else {
			Result = developVALink(link, word.getVAPast());
		}

	}
	return Result;
}

// Develops a Muscle link of Bone to Spine
public String developVALink(int linkType, String linkValue) {
	String VALink = null;
	switch (linkType) {
```

```
                case 2:
                      VALink = linkValue + "가 ";
                      break;
…….
……..
……... further Muscle Link cases omitted

                default:
                      VALink = "-df-";
                      break;

                }
                return VALink;
        }
…….
……..
……... further methods omitted

}
```

8.2.6 Translate (Procedure)

```
public class JKTranslate {
      private String[] Output;

      private String[] BonesOutput;

      private String[] FinalOutput;

      private RuleGate RG;

      public JKTranslate(RuleGate rule) {
            FinalOutput = new String[2];
            RG = rule;
      }

      public String[] translate(Word[][] WR, GrammarSkeleton[] GSR) {

            for (int count = 0; count < 2; count++) {
                  BonesOutput = translateBones(WR, GSR[count].getBoneIndices(),
                              count, GSR[count].getMuscleIndex());
                  int SpineIndex = GSR[count].getSpineIndex();
                  switch (SpineIndex) {
                  case 1: // "－から－。"
                        System.out.println("spine 1 chosen");
                        FinalOutput[count] = BonesOutput[0] + " 때문에 " + BonesOutput[1];
                        System.out.println();
                        break;
.....
.....
........ further Spine cases omitted

                  default:
                        System.out.println("default spine chosen");
                        FinalOutput[count] = "dfs";
                        break;
                  }
```

```
        }
        return FinalOutput;
}

public String[] translateBones(Word[][] WR, int[] BGI, int sentNum,
              String mIndex) {
        int click = 0;
        BonesOutput = new String[2];
        for (int count = 0; count < 2; count++) {

              int BoneIndex = BGI[count];

              switch (BoneIndex) {
              case 1: // "N-はV-"
                    System.out.println("bone 1 chosen");
                    Output[sentNum] = RG.unOrNun(WR[sentNum][click]);
                    Output[sentNum] = Output[sentNum]
                                + RG.verbEnd(WR[sentNum][click + 1], Integer
                                            .valueOf(mIndex.charAt(count)));
                    click = click + 2;
                    System.out.println(Output[sentNum]);
                    break;
              case 2: // "N-がV-"
                    System.out.println("bone 2 chosen");
                    Output[sentNum] = RG.iOrGa(WR[sentNum][click]);
                    Output[sentNum] = Output[sentNum]
                                + RG.verbEnd(WR[sentNum][click + 1], Integer
                                            .valueOf(mIndex.charAt(count)));
                    click = click + 2;
                    System.out.println(Output[sentNum]);
                    break;
              case 3: // "N-にV-"
                    System.out.println("bone 3 chosen");
                    Output[sentNum] = WR[sentNum][click].getKorWord() + "에";
                    Output[sentNum] = Output[sentNum]
                                + RG.verbEnd(WR[sentNum][click + 1], Integer
                                            .valueOf(mIndex.charAt(count)));
```

```
            click = click + 2;
            System.out.println(Output[sentNum]);
            break;
        case 4: // "N-でV-"
            System.out.println("bone 4 chosen");
            Output[sentNum] = RG.eseoOrRo(WR[sentNum][click]);
            Output[sentNum] = Output[sentNum]
                        + RG.verbEnd(WR[sentNum][click + 1], Integer
                                    .valueOf(mIndex.charAt(count)));
            System.out.println(Output[sentNum]);
            break;
        case 5: // "N-はA-"
            System.out.println("bone 5 chosen");
            Output[sentNum] = RG.unOrNun(WR[sentNum][click]);
            Output[sentNum] = Output[sentNum]
                        + RG.adjEnd(WR[sentNum][click + 1], Integer
                                    .valueOf(mIndex.charAt(count)));
            click = click + 2;
            System.out.println(Output[sentNum]);
            break;
        case 6: // "N-がA-"
            System.out.println("bone 6 chosen");
            Output[sentNum] = RG.iOrGa(WR[sentNum][click]);
            Output[sentNum] = Output[sentNum]
                        + RG.adjEnd(WR[sentNum][click + 1], Integer
                                    .valueOf(mIndex.charAt(count)));
            click = click + 2;
            System.out.println(Output[sentNum]);
            break;
        case 7: // "V-N-はA-"
            System.out.println("bone 7 chosen");
            Output[sentNum] = WR[sentNum][click].getKorWord();
            Output[sentNum] = Output[sentNum]
                        + RG.unOrNun(WR[sentNum][click + 1]);
            Output[sentNum] = Output[sentNum]
                        + RG.adjEnd(WR[sentNum][click + 2], Integer
                                    .valueOf(mIndex.charAt(count)));
```

```
                        click = click + 3;
                        System.out.println(Output[sentNum]);
                        break;
…….
……..
……... further Bone cases omitted

                  default:
                        System.out.println("default bone chosen");
                        Output[sentNum] = "dfb";
                        break;
                  }
            }
            return Output;
      }

}
```

8.3 FBCT Access Algorithm (Skeletal Version)

This algorithm is scripted in J2ME and is the skeletal version that is compiled and deployed onto any J2ME supporting cellular phone to give the user access the FBCT translation service. The skeletal version has no extensive interface structure, user authentication measures or options including add-on features, it simply performs text communication with the FBCT servlet that resides on the server. Two classes exist, however only *JKTranslation* is predefined as a MIDlet that has a JAD (Java Application Descriptor) file and a JAR (Java Application Resource) file associated with it, making it a selectable J2ME program on the cellular phone. Conversely, the *HttpHelperConnection* class exists only as a support class which has no JAD or JAR file associated with it.

8.3.1 JKTranslation Class

```
import java.io.*;
import java.lang.*;
import javax.microedition.io.*;
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;


public class JKTranslation extends MIDlet implements CommandListener {


      private static String url = "http://www.workushard.co.nz/JKT";

      private Display display;

      private Command exitCommand = new Command("Exit", Command.EXIT, 2);

      private Command okCommand = new Command("OK", Command.OK, 1);

      private Command backCommand = new Command("Back", Command.BACK, 1);

      private Command selectCommand = new Command("Select", Command.OK, 1);

      private Command translateCommand = new Command("Translate", Command.OK, 1);

      private String[] modes = { "Japanese >> Korean Translation" };

      public TextBox entryForm;

      public List mainMenu;


      public JKTranslation() {}

      protected void destroyApp(boolean unconditional)
                  throws MIDletStateChangeException {
```

```
        exitMIDlet();
    }

    protected void pauseApp() {
    }

    protected void startApp() throws MIDletStateChangeException {
        if (display == null) {
            // first time called...

            initMIDlet();
        }
    }

    private void initMIDlet() {
        display = Display.getDisplay(this);
        mainMenu = new MainMenu();
        display.setCurrent(mainMenu);
    }

    public void exitMIDlet() {
        notifyDestroyed();
    }

    public void commandAction(Command c, Displayable d) {
        if (c == translateCommand) {
            StatusForm f = new StatusForm(entryForm.getString());
            display.setCurrent(f);
            f.start();
        } else if (c == okCommand) {
            display.setCurrent(entryForm);
        } else if ((d == mainMenu)
                & ((c == List.SELECT_COMMAND) | (c == selectCommand))) {
            switch (mainMenu.getSelectedIndex()) {
            case 0: // JK Translate
                entryForm = new EntryForm();
                display.setCurrent(entryForm);
                break;
```

```
                default:
                }
        } else if (c == backCommand) {
                if (display.getCurrent() == entryForm) {
                        display.setCurrent(mainMenu);
                } else {
                        exitMIDlet();
                }
        } else if (c == exitCommand) {
                exitMIDlet();
        } else {
                exitMIDlet();
        }
}


// The main menu

class MainMenu extends List {
        MainMenu() {
                super("Mode Selection", Choice.IMPLICIT, modes, null);
                Ticker mainMenuTicker = new Ticker(
                                "Select the mode of translation you wish to use.");
                setTicker(mainMenuTicker);
                addCommand(exitCommand);
                addCommand(selectCommand);
                setCommandListener(JKTranslation.this);

        }
}

// The text entry form

class EntryForm extends TextBox {
        EntryForm() {
                super("Text Capture", "", 80, 0);
                Ticker entryFormTicker = new Ticker(
                                "Enter the Japanese text you wish to be translated into Korean.");
```

```
            setTicker(entryFormTicker);
            addCommand(exitCommand);
            addCommand(backCommand);
            addCommand(translateCommand);
            setCommandListener(JKTranslation.this);
      }
}

// A status for for displaying messages as the
// data is sent

class StatusForm extends Form implements Runnable,
            HttpConnectionHelper.Callback {
      StatusForm(String text) {
            super("Status");

            // Convert the string into a byte array.
            // Doing it this way ensures that the
            // characters retain their encoding.

            try {
                  ByteArrayOutputStream bout = new ByteArrayOutputStream();
                  DataOutputStream dout = new DataOutputStream(bout);

                  dout.writeUTF(text);
                  data = bout.toByteArray();

                  dout.close();
            } catch (IOException e) {
            }
      }

      // Updates the display.

      void display(String text) {
            if (message == null) {
                  message = new StringItem(null, text);
                  append(message);
```

```
        } else {
            message.setText(text);
        }
}

// Done.

void done(String msg) {
        display(msg != null ? msg : "Done.");
        addCommand(okCommand);
        setCommandListener(JKTranslation.this);
}

// Callback for making the HTTP connection.

public void prepareRequest(String originalURL, HttpConnection conn)
            throws IOException {
        conn.setRequestMethod(HttpConnection.POST);
        conn.setRequestProperty("User-Agent",
                    "Profile/MIDP-1.0 Configuration/CLDC-1.1");
        conn.setRequestProperty("Content-Language", "en-US");
        conn.setRequestProperty("Accept", "application/octet-stream");
        conn.setRequestProperty("Connection", "close");
        conn.setRequestProperty("Content-Length", Integer
                    .toString(data.length));

        OutputStream os = conn.openOutputStream();
        os.write(data);
        os.close();
}

// Do the connection on a separate thread to
// keep the UI responsive

public void run() {
        HttpConnection conn = null;

        display("Obtaining HttpConnection object...");
```

```
        try {
              conn = HttpConnectionHelper.connect(url, this);

              display("Connecting to the server...");
              int rc = conn.getResponseCode();

              if (rc == HttpConnection.HTTP_OK) {
                    StringBuffer text = new StringBuffer();

                    // Here's where you read the data.
                    // This case expects an integer
                    // followed by zero or more
                    // strings.

                    try {
                          DataInputStream din = new DataInputStream(conn
                                      .openInputStream());

                          int n = din.readInt();
                          while (n-- > 0) {
                                text.append(din.readUTF());
                                text.append('\n');
                          }
                    } catch (IOException e) {
                    }

                    done("Response is:\n" + text.toString());
              } else {
                    done("Unexpected return code: " + rc);
              }
        } catch (IOException e) {
              done("Exception " + e + " trying to connect.");
        }
}
```

```
            // Starts the upload in the background

            void start() {
                  display("Starting...");

                  Thread t = new Thread(this);
                  try {
                        t.start();
                  } catch (Exception e) {
                        done("Exception " + e + " trying to start thread.");
                  }
            }

            private StringItem message;

            private byte[] data;
      }
}
```

8.3.2 HttpConnectionHelper Class

```
import java.io.*;
import javax.microedition.io.*;

  public class HttpConnectionHelper {

    public interface Callback {
      void prepareRequest( String originalURL,
                           HttpConnection conn )
           throws IOException;
    }

    public static HttpConnection connect( String url )
                                    throws IOException {
      return connect( url, null );
    }

    public static HttpConnection connect(
            String url, Callback callback )
                         throws IOException {
      HttpConnection conn = null;
      String         originalURL = url;

      while( url != null ){
                          Connector.open( url );

        if( callback != null ){
          callback.prepareRequest( originalURL,
                                   conn );
        }

        int rc = conn.getResponseCode();

        switch( rc ){
            case HttpConnection.HTTP_MOVED_PERM:
            case HttpConnection.HTTP_MOVED_TEMP:
            case HttpConnection.HTTP_SEE_OTHER:
```

```
            case HttpConnection.HTTP_TEMP_REDIRECT:
              url = conn.getHeaderField( "Location" );
              if( url != null && url.startsWith(
                                              "/*" ) ){
                StringBuffer b = new StringBuffer();
                b.append( "http://" );
                b.append( conn.getHost() );
                b.append( ':' );
                b.append( conn.getPort() );
                b.append( url );
                url = b.toString();
              }
              conn.close();
              break;
            default:
              url = null;
              break;
          }
        }

        return conn;
    }
}
```