Text entry for mobile devices

Sanju Sunny

2005

Sanju, S. (2005). Text entry for mobile devices. Master's thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/2440>

<https://doi.org/10.32657/10356/2440>

Nanyang Technological University

Downloaded on 26 Apr 2023 21:11:11 SGT

TEXT ENTRY FOR MOBILE DEVICES

SANJU SUNNY

SCHOOL OF COMPUTER ENGINEERING
NANYANG TECHNOLOGICAL UNIVERSITY

2005

TK
6570
•M6
S227
2005

Text Entry for Mobile Devices

SANJU SUNNY

School of Computer Engineering

**A thesis submitted to Nanyang Technological University
in fulfillment of the requirement for the degree of
Master of Engineering**

2005

ACKNOWLEDGEMENTS

There are many people who deserve a tremendous amount of thanks and credit for their help along the way. First and foremost, I would like to thank my Project Supervisor, Dr. Yow Kin Choong, for his unfailing support and guidance throughout the course of this project. His constant encouragement has been critical to my success.

I am thankful to Dr. Mark Ryan for sharing his research ideas in the early phase of the project. I am also grateful to Dr. Theng Yin Leng, for giving me her views on the HCI aspects of the project. Thanks are due to Dr. Paul Graham Doyle and Dr. Min-Yen Kan for providing the corpora used in the design of Glyph. I would also like to express my gratitude to the laboratory technicians of the Database Technology Laboratory for providing me with a productive working environment.

Last but not the least, I would like to thank my friends for their critical reviews on the project and my family for their constant support.

TABLE OF CONTENTS

Abstract	i
Acknowledgements	ii
Table of Contents.....	iii
List of Figures.....	v
List of Tables.....	vii
Chapter 1: Introduction	
1.1 Background.....	01
1.2 Motivation	02
1.3 Contributions of the Project.....	04
1.4 Structure of the Report	06
Chapter 2: Text Entry on Mobile Phones : Theory and Analysis	
2.1 Text Entry for Mobile Phones.....	08
2.2 Predictive Text Systems.....	10
2.3 Predictive Systems and the User	16
2.4 Models for Mobile Text Entry	17
Chapter 3: Evaluation of Current Mobile Text Entry Systems	
3.1 Key Based Systems.....	21
3.2 Intelligent Systems.....	23
3.3 Other Systems.....	27
Chapter 4: The Prototyping Framework: Design and Implementation	
4.1 Object Oriented Application Frameworks.....	30
4.2 Framework Development Approaches.....	32
4.3 Framework Design Elements	33
4.4 Framework Design	35
4.5 Application Development using the Framework.....	41
4.6 Evaluating the effectiveness of the Framework	43
Chapter 5: The Design of Glyph : An Efficient and Adaptive Text Entry System...	
5.1 Overview	47
5.2 The Nature of Mobile Text Messaging.....	47
5.3 Data sources for the construction and testing of language models...	50
5.4 Inferences from Corpora	47
5.5 n-gram Analysis.....	59

- 5.6 Limitations of Corpora Analysis 63
- 5.7 Design Goals for Glyph 63
- 5.8 Glyph Architecture..... 65
- 5.9 Techniques Implemented to Improve Text Entry 69
- Chapter 6: **Implementation**
 - 6.1 Prototype Development..... 74
 - 6.2 System Architecture 74
 - 6.3 Generating Predictions 80
 - 6.4 Displaying Predictions 82
 - 6.5 Learning 83
 - 6.6 System Setup 83
- Chapter 7: **Evaluating Glyph**
 - 7.1 MultiTap..... 86
 - 7.2 T9 87
 - 7.3 Glyph..... 87
 - 7.4 KSPC Comparison..... 89
 - 7.5 Improvements in Text Entry 92
 - 7.5 Issues in Adaptation and Optimization 94
- Chapter 8: **Conclusion**
 - 8.1 Conclusion..... 97
 - 8.2 Contributions of the Project..... 98
 - 8.3 Recommendations for Future Work 98
- References
- Appendix A : The Research Journal : Design and Implementation.....
- Appendix B : Calculation of Degree of Disambiguity

LIST OF FIGURES

Figure 2.1. The Standard mobile phone	09
Figure 3.1. Classification of Text Entry Systems.....	19
Figure 3.2. MultiTap.....	21
Figure 3.3. LetterWise	23
Figure 3.4. T9.....	24
Figure 3.5. EziText	26
Figure 3.6. The FastTap Keypad	27
Figure 3.7. The Q12 Keypad.....	27
Figure 3.8. ThumbScript	28
Figure 4.1. The decomposition of an application	31
Figure 4.2. The iterative framework development process	32
Figure 4.3. Inversion of control in a Framework.....	35
Figure 4.4. Core Classes	36
Figure 4.5. Text Processing.....	38
Figure 4.6. Layout.xml.....	39
Figure 4.7. Startup Sequence	40
Figure 4.8. Building a Custom Device Interface	41
Figure 4.9. A T9 implementation using the Framework	42
Figure 4.10. Minimum Code to Implement MultiTap using the Framework	45
Figure 5.1. Character frequencies in corpora	55
Figure 5.2. Degree of Disambiguity.....	57
Figure 5.3. Character frequencies for key	58
Figure 5.4. A set of messages created by user P2.....	59
Figure 5.5. DoA for unigram and digram models.....	61
Figure 5.6. Best character match for key D6	63
Figure 5.7. Message Composition using Predictive Text Systems	66
Figure 5.8. Long Term Memory Structure.....	68
Figure 5.9. Crossing Word Boundaries.....	69
Figure 6.1. System Architecture	75
Figure 6.2. Glyph Data Analysis Tools	76

Figure 6.3. Trie Management Tools 77

Figure 6.4. The node data structure..... 78

Figure 6.5. Model Construction 79

Figure 6.6. The DataManager class 80

Figure 6.7. Predictions in Glyph 81

Figure 6.8. Remapped keyboard..... 84

Figure 6.9. Deploying Glyph 84

Figure 7.1. Comparing KSPC..... 89

Figure 7.2. Testing with different data sources 90

Figure 7.3. KSPC with word prediction..... 91

Figure 7.4. The effect of word prediction 91

Figure 7.5. The tradeoff between KSPC and model size 92

Figure 7.6. Entering personal lingo in Glyph..... 93

Figure 7.7. Side effects of learning 94

Figure 7.8. Predictions across the word boundary for order 3 and 4 n-grams 95

LIST OF TABLES

Table 5.1. Type of shortened word used in messages..... 49

Table 5.2. Corpora used by the project..... 50

Table 5.3. Ten most frequent words in each corpora 53

Table 5.4. Ten most frequent characters in each corpora 54

Table 5.5. Most frequent words not present in the dictionary..... 56

Table 5.6. Characters mapped to each key..... 57

Table 5.7. Messaging characteristics for each user..... 59

Table 5.8. Most frequent words used by P1 and P2 59

Table 5.9. KSPC for MultiTap for each corpus 60

Table 5.10. Effect of Model Character Count on Memory Footprint..... 70

Table 5.11. Effect on Context Length on Memory Footprint 70

A B S T R A C T

The mobile phone has become the most widely used communication medium in the world creating a device that is almost an extension of the self. However, several issues must be overcome before the true potential of this medium is realized.

The first section of this report describes the conceptualization, design and development of a prototyping framework for text entry research on mobile devices. The primary goal of the framework is to help mobile text entry researchers produce high-fidelity working prototypes of their concept. It aims to ease development effort and maximize reusability of code. The framework is adaptable to handle innovative device designs and sufficiently flexible in its design to enable a developer to customize interactions specifically for his project.

The primary focus of this thesis is the design of Glyph: An efficient and adaptive text entry system for mobile phones. Glyph is a learning system that transparently adapts to the user. The core of the system is a character-based predictor augmented with a small dictionary for added word completion capabilities. In tests, Glyph has been found to require less than 1 key stroke per character, significantly less than that for established systems. Several strategies have been described to improve the text entry experience. However, user evaluations are essential to measure real world text entry rates and to study the cognitive loads incurred. This thesis describes the design, implementation and evaluation of Glyph. It also provides an analysis of the nature of mobile text messaging inferred from mining multiple data sources.

Chapter 1

Introduction

“It is as if the mobile phone has come to meet a biological need; more, it has become part of our bodies, and therefore of ourselves.” [Hulme et.al, 2001]

1.1 Background

If any further evidence is needed that technology can change the world - and often in totally unanticipated ways - consider what happened in the Philippines in January 2001. Thousands of Filipinos, unhappy with their corrupt government, took to the streets to demonstrate against President Joseph Estrada, ultimately forcing him to resign. Mobile phones played a key role in stimulating and organizing the protests - not voice communications, but short text messages sent from one phone to another or from one phone to many others [Teich, 2001]. First they were used to send political jokes; later they spread the word on where demonstrations were being held.

The mobile phone has become the most widely used communication medium in the world, which is making our lives simpler by delivering services into our palms. There are far more mobile handsets than the number of fixed lines and the mobile penetration rate is much higher than that of the Internet [Rheingold, 2002]. Consumers are becoming more and more comfortable with the use of their mobile phones as a device for communicating with their friends, colleagues and family. Services such as sports results, betting games and stock market news sent directly to mobile phones and a better quality of hand-sets (color, large displays and accessories) have helped fuel the growth in texting. Text messaging is seen as a medium of choice - being simple, cost effective, instant, discrete, fashionable and transparent to use with ubiquitous coverage.

The phrase ‘*txt msging*’ might not qualify as well formed English but that does not make it any less readable to today’s youth. The evolving language of text messaging characterized by emoticons, abbreviations and homonyms is in part the result of a

community of users trying to overcome the obvious flaws of that ubiquitous device, the mobile phone.

Over 70% of mobile phone users now use their handsets for text messaging. Sources in Singapore report that about three-quarters of Singapore's 4 million inhabitants subscribe to mobile phone services and every day, more than half send text messages [MobileYouth, 2002]. Worldwide, the figures run into billions, despite a lack of proactive marketing by service providers. However this phenomenal messaging rate is in spite of and not because of the usability of the mobile devices. The scope exists then, to eclipse even these figures as new inroads are made into making text entry on mobile devices more usable, enjoyable and efficient.

1.2 Motivation

A major trend in the recent past has been a move to ever smaller, ever more portable computing devices. Computers have become smaller over the years, shrinking from the mainframe of 30 years ago to the laptop of today. Technological and market forces have pushed today's computing devices even beyond the size of a laptop. Mobile phones have begun to encompass much more than simply phone calls, allowing email, access to the web, and text communication. A new market has also emerged for so-called personal digital assistants or PDA. These PDA, first popularized by the Apple Newton, and later driven by other, even smaller devices, offer features like web browsing, document creation, appointment management, and note taking.

A common requirement of many of these applications is the need for entering text, whether it is to compose an email or create a document on the move. The recent inclusion of instant messaging capabilities on mobile devices has only served to increase the strain on current text entry methods as they have to cope with the additional requirement of being effortless and *fast*.

When users are not forced to use interactive systems in a work context but choose to use them to achieve personal goals, excellent user performance is not enough. These interactive systems have to satisfy an existing user need (be useful) and they have to be fun and pleasant to use (user acceptance).

1.2.1 Small Size, Big Problems

Any text entry method for mobile devices has to overcome some fundamental issues. The key factor at design time is the size of the device. Designing for small devices creates obvious challenges for HCI researchers [Dunlop et.al, 2002]. There is a limit on acceptable size that is dictated by user perception on how bulky a mobile device can be before it loses its 'mobility'. Unlike technological barriers which can be overcome, user perception is more difficult to work around.

The QWERTY keyboard, while ubiquitous on the desktop, is ill-suited for use on mobile devices owing to its size. Screen size is at a premium and much research has been expended on making optimal use of the available visible area. There is an increasing trend among manufacturers to increase the screen size of the devices. The overall size of the device remaining a constant, this translates into smaller areas for input devices or adoption of non-button based input like handwriting recognition.

1.2.2 Text Input Capability

The typical input capabilities of a mobile phone include 12 buttons for data entry and phone specific input methods for control and navigation that include buttons, joysticks, scroll wheels etc. The 12 keys include the numbers 0-9 and two keys for '#' and '*'. The alphabets from 'a' to 'z' are spread across these keys using the international Standard Ambiguous Code (SAC) [ISO, 1994]. Since there are fewer keys than there are alphabets, this is an ambiguous arrangement with each key mapped to two or three characters. Depending on the text entry method being used, the user typically has to exert additional physical and/or mental effort to enter text owing to the need to disambiguate characters or words. This is the subject for a major body of research.

1.2.3 Developmental Issues

Designing a new text entry method is a challenge on many fronts. First and foremost, it is a labor intensive process. It is also expensive, since a working prototype must be built, and then tested with real users [MacKenzie, 2002A].

The effectiveness of many computing systems can be proven with objective tests and quantifiable results. Text entry methods however interface directly and intimately with the end user. Subjective satisfaction is important because if users do not like the newer

technologies of text entry, they are less likely to buy a phone which offers that technology. Users prefer to stick with things that are more familiar to them, that are available on older phones and that they already know how to use. Introducing new methods of text entry often causes apprehension and mistrust before the users actually use the technology. An empirical evaluation with users is therefore, paramount to the viability of new methods of text data entry. This is an expensive and time consuming task.

1.2.4 Alternate Input Methods

The ease of use of Palm's handwriting recognition software, Graffiti, has long being touted as one of the key selling points of the Palm Pilot line of devices. While not perfect, the use of handwriting recognition in phones is picking up as can be seen by its adoption in recent phones. This also allows for more screen space as the keypad is replaced by a 'soft' or virtual keyboard. Many of the current crop of text entry systems support stylus based input.

Voice recognition has been touted as the next big thing. However, for all practical purposes, it has not lived up to its promise. Voice recognizers that are considered to be close to the state-of-the-art today do not work well enough with all speakers [Barber, 1997]. Voice technology is expensive, both economically and computationally and at its current stage of development is not accurate enough for general text entry. It is obtrusive, insecure, sensitive to noise and, of course, not private. The cognitive load of voice interfaces is greater than hand/eye co-ordination. Speaking requires the user's short-term memory, whereas hand/eye co-ordination is parallel processed in other parts of the brain. Voice input in current generation phones is limited to simple address book lookups rather than full-fledged text entry.

1.3 Contributions of the Project

The major challenges in text entry for mobile phones have been highlighted in the previous section. This project addresses two particular issues, namely that of predictive text systems and prototyping for text entry research.

1.3.1 A Prototyping Framework for Text Entry Research

The first contribution of this project is the conceptualization, design and development of a prototyping framework for text entry research on mobile devices. The primary goal of the framework is to help text entry researchers produce high-fidelity working prototypes of their concept. It aims to ease development effort and maximize reusability of code. The framework is adaptable to handle innovative device designs and sufficiently flexible in its design to enable a developer to customize interactions specifically for his project.

The framework is expected to be useful for prototyping research projects that are concerned with

- Language modeling and optimization for improving text entry
- Predictive text systems
- Innovative keypad design
- Models for text entry
- Usability of mobile devices and services
- Social impact of text messaging

An implementation of the framework won the Runners up Award at the Encentuate Innovation Competition 2003. A paper on the framework titled “A Prototyping Framework for Mobile Text Entry Research” was presented at the 6th Asia Pacific Conference on Computer Human Interaction (APCHI 2004) held in Rotorua, New Zealand.

1.3.2 Glyph: An Efficient and Adaptive Text Entry System for Mobile Phones

The major contribution of this project is Glyph; A predictive text entry system for mobile phones. Glyph is a learning system that transparently adapts to the user. The adaptive model of Glyph sidesteps the learning problem in current text entry systems by removing the need for text entry mode switches in learning, thus avoiding the heavy cognitive loads associated with explicit learning models. In addition, several enhancements are proposed to enhance text entry rate. The core of the system is a character-based predictor augmented with a small dictionary for added word-completion capabilities.

This report describes the design of Glyph using the prototyping framework, its implementation for Smartphones and an evaluation comparing it with established text entry systems. The report also contributes an extensive survey of current text entry systems for mobile phones critiquing both software and hardware solutions. It also includes an analysis of the nature of mobile text messaging and a discussion on how it affects the design of new text entry methods.

A paper on Glyph has been submitted to MobiCom 2005, the Annual International Conference on Mobile Computing and Networking.

1.4 Structure of the Report

This report is organized into 7 chapters by the nature of the content.

Chapter 1 has given a brief overview of the challenges facing text entry researchers, discussed the motivations driving the project objectives and defined the contributions of the project.

Chapter 2 explores the subject of mobile text entry in further detail. Predictive text entry systems and the theoretical background behind their operation are introduced.

Chapter 3 provides an overview of current systems for text entry on mobile phones. Systems are critiqued and compared on the basis of various metrics. Beyond helping with the design of Glyph, the case studies also form the domain analysis research critical to the design of the prototyping framework.

Chapter 4 discusses the design and implementation of the prototyping framework. It elaborates on the design of object-oriented application frameworks and the concept of Design Patterns. Directions for future improvements are suggested as is an evaluation of the framework.

Chapter 5 outlines the design of Glyph. This chapter details the various data sources used to create the data model of Glyph and describes the inferences obtained from mining this data. It details the working of the predictive core and elaborates on key design challenges and their solutions.

The system architecture of Glyph and its implementation is described in Chapter 6. It provides an overview of the runtime behavior of the text entry system. The construction of key components of the system are outlined.

Chapter 7 provides an evaluation of Glyph comparing it with established text entry systems. It includes both quantitative and qualitative evaluations.

Chapter 8 concludes the report with a review of the work and the conclusions obtained. Several recommendations for improving the text entry experience are proposed.

Appendix A provides a brief overview of the Research Journal software developed to streamline the project research and to download and organize publications.

Chapter 2

Text Entry on Mobile Phones: Theory and Analysis

2.1 Text Entry for Mobile Phones

There are two competing paradigms for mobile text input: pen-based input and keyboard-based input. Both emerged from ancient technologies ("ancient" in that they predate computers): typing and handwriting. User experience with typing and handwriting greatly influences expectations for text entry in mobile computing; however the two tasks are fundamentally different.

A key feature of keyboard-based text entry is that it directly produces machine-readable text (i.e., ASCII characters), a necessary feature for indexing, searching, and handling by contemporary character-based technology. Handwriting without character recognition produces "digital ink". This is fine for some applications such as annotation, visual art, and graphic design. However, digital ink requires more memory and in general it is not well managed by computing technology. For handwritten text entry to achieve wide appeal, it must be coupled with recognition technology.

An important consideration implicit in the discussion of text input technology is user satisfaction. Users' expectations for text entry are set by current practice. Touch typing speeds in the range of 20 - 40 words per minute (wpm) are modest and achievable for hunt-and-peck typists. Rates in the 40 - 60 wpm range are achievable for touch typists, and with practice, skilled touch typists can achieve rates greater than 60 wpm. Handwriting speeds are commonly in the 15 - 25 wpm range. These statistics were collected from [MacKenzie, 2002a].

Figure 2.1. The Standard mobile phone

The project focuses on building a system targeted at making maximum use of and at the same time overcoming the idiosyncrasies of the standard mobile phone keypad.

The standard North American letter/number assignment found on telephones in Canada and the USA since beginning of the 20th century did not feature the Q and Z alphabets. The assignment of numbers to letters predates Dual Tone Multi Frequency (DTMF) and touch dialing and is the same as is found on many old rotary dial phones. Literally all companies building telephones for consumers on that continent used this assignment.

The second half of the 20th century saw the demise of exchange names and the relaxation of rules on what digits could appear in area codes and exchange prefixes. This, combined with the use of touch tone dialing finally led the telecom industry to come up with places for the remaining letters. The natural course of progression was to keep the existing phone key pads and dials and to find a place for the letters "Q" and "Z" that were

consistent with the classic keypad. The current standard for mobile keypad layout is the ISO/IEC 9995-8:1994 [ISO, 1994] (Figure 2.1).

The 12-key keypad consists of number keys 0-9 and two additional keys (# and *). Characters A-Z are spread over keys 2-9 in alphabetic order. Since there are fewer keys than the 26 needed for the characters A-Z, three or four characters are grouped on each key. Overloading multiple letters on a single key creates potential ambiguity as to which letter was intended. For example, if the user presses key 2, the system must determine which of the characters A, B, or C the user intends. For applications involving a simple, predetermined vocabulary, a simple hashing scheme will suffice. However, the problem becomes more complex when we need to deal with arbitrary text. In practice, a key on the mobile phone can be mapped to over 10 unique characters.

Disambiguation methods can be classified into two types [Davis, 1991]. In *explicit* disambiguation, the user is expected to perform extra actions to indicate to the system which letter is intended. In some cases, it is possible to use *implicit* disambiguation, where the system figures out for itself which letter is intended.

2.2 Predictive Text Systems

The goal of a predictive text system is to increase the efficiency of text entry (e.g.: keystroke reduction) by predicting user input. The predictions can be based on many sources of information including information about previous words, current context, and/or current letters.

These predictions can be used in many useful ways. Predictive systems can be used as a technique for implicit disambiguation. It can also provide word completion in which the system suggests candidate completions for the word the user intends to type. Such systems usually required a portion of the word intended or a prefix/word stem to make their prediction. There are many kinds of predictive text systems. *Word Prediction* systems try to guess the next word while *Character Prediction* systems try to guess the next character. Word prediction has been defined as offering the user a list of words after a word has been typed or selected, based on previous words rather than on the basis of letters [Cook et al, 1995]. This definition combines classical word prediction with word completion. The advantages of increased predictive accuracy are not limited to the

keystroke savings. By providing orthographic and grammatical cues, effective word prediction can improve the quality (as well as the quantity) of message production. However, the reverse can be said to be true for mobile messaging; word prediction works to constrain the richness afforded by the communication medium.

While research in mobile text entry is a fairly recent development, a vast body of work dating back to the 1960s exists for the field of Augmentative and Alternative Communication (AAC). AAC is the field of study concerned with providing devices or techniques to augment the communication ability of a person whose disability makes it difficult to communicate in an understandable manner [Wester, 2003]. We argue that the mobile user of today faces a challenge in his ability to communicate due to the low usability of current text entry methods. He is for all practical purposes disabled; usually limited to the use of just two fingers to communicate through text messages. This realization allows us to leverage on AAC research to improve current text entry methods. Word prediction is a technique frequently used in the AAC field of writing support devices with the purpose of improving the keyboard text input speed, and the quality of spelling and syntax [Copestake, 1997].

2.2.1 Information-Theoretic Bounds

Information theory provides a theoretical explanation to why predictive text systems work in the context of the project. Formally, an information source S with a repository of possible messages M_k , $1 \leq k \leq n$ selects one message for transmission. Information theory defines the information content or source entropy $H(S)$ of a set of messages in terms of the probability $P(M_i)$ that this message is chosen by the source, as the average value of the information associated with each of the messages [Shannon, 1948]:

$$H(S) = - \sum_{i=1 \dots n} P(M_i) \log_2 P(M_i) \quad (1)$$

The source entropy denotes the average information content per message, in units of bits and reaches its maximum value when each message is equally likely.

In the context of mobile text entry, the message source is the English language. For the sake of clarity in the discussion, we exclude punctuation and other extraneous symbols. If we assume that each of the 26 alphabets can occur in text with equal probability, the

entropy of English is $\log_2(26) = 4.7$ bits per character. However, Shannon has identified a fair amount of redundancy in the English language [Shannon, 1950] (for example, the letter 'u' always follows a 'q'). English does not use all letters with equal probability. Better estimates of the entropy of English are obtained by considering entire words or sentences rather than just individual characters. In his experiments on full English text, Shannon obtains estimates as low as 0.6 to 1.3 bits per letter.

On the other hand, the maximum information that can be created by the mobile device (10 keys) is $\log_2(10) = 3.32$ bits per keystroke. However, the standard keypad layout is not an optimal mapping and this reduces entropy to approximately 3.07 bits per letter [Rau et al, 1996]. Despite this reduction, it can be seen that this figure is higher than Shannon's estimate of the entropy of English. Thus, in principle it is possible to disambiguate user input with the mobile phone keypad by exploiting statistical and other linguistic constraints.

2.2.2 Prediction Techniques

Statistical Word Prediction

Language statistics are a useful source of information for word prediction systems. Statistical word prediction techniques use the current sentence context to provide meaningful predictions for the following word.

In n-gram word prediction methods, the previous n-1 words are used to predict the current (n*) word. The n-gram data is collected by counting the occurrence of each unique n word sequence in a large corpus called the training text. For augmentative communication applications, n-gram techniques have been limited to unigram (n=1) and bigram (n=2) word prediction, although trigram (n=3) and higher n-gram orders are commonly used in other language-related fields such as speech recognition and machine translation. For n-gram orders higher than unigrams (n>1), there is such a large number of linguistically valid n word sequences that even in extensive training texts, some sequences will not appear, or will occur too infrequently to provide statistically meaningful data. A prediction system must therefore temper its higher order n-gram predictions with lower order, more reliable, n-gram predictions. This is generally done through a linear interpolation process wherein predictions from each n-gram order are

weighted by a different factor. Even when using this compensatory method, however, the effectiveness of an n-gram prediction model is highly dependent upon the size of the training text.

Leshner et al. investigated the effects of n-gram order and training text size on word prediction [Leshner et al, 1999]. They found that predictive performance can be improved by using higher order n-gram prediction techniques and larger training texts. Switching from unigram to trigram prediction increased keystroke savings by 7.1 percentage points (at 3 million words), while tripling the training text size from 1 million to 3 million words improved savings by nearly 2.5 percentage points (using trigram prediction).

For unigram prediction, performance begins to asymptote at 3 million words - the system has learned nearly all it can about how frequently individual words appear. For bigram and trigram prediction, however, performance continues to improve at 3 million words - because there are so many linguistically valid two and three word sequences, the system has yet to derive meaningful statistics for most of them. These results imply that substantial gains may be realized by utilizing even larger training texts.

n-gram methods are relatively simple to incorporate into an application. However, the cost of the performance improvement associated with n-gram methods is reflected in the extended memory requirements for the resulting statistical databases. Further research is required into pruning higher order n-gram data structures before they can be used efficiently on resource limited mobile devices.

In an attempt to establish that word prediction performance has not reached a fundamental limit, Leshner et al. performed an experiment designed to measure the performance of human word predictors [Leshner et al, 2002]. They found that the average human subject could outperform statistical techniques by a significant margin, and that the best human subject could outperform the automated methods by 10 percentage points suggesting that there is ample room for improvement in machine word prediction.

Statistical Letter Prediction

Letter prediction systems are usually found in tandem with ambiguous input devices where multiple symbols are mapped to the same key. Although a substantial body of research has been devoted to improving word prediction methods, there have been few

investigations into more effective character prediction techniques, despite the important role that accurate estimation of letter probabilities can play in augmenting communication. Research indicates that character prediction can provide better keystroke savings than can word prediction in certain systems - over 30% in some cases [Lesh et al, 1998].

Character prediction systems rely upon a database of inter-character statistics, generated by analyzing a large corpus of representative text. In *n*-gram prediction, the past *n*-1 characters are used to predict the current (*n*th) character. In *k*-gram prediction, the first *k*-1 characters of the current word are used to predict the current (*k*th) character. In general, *k*-gram character prediction is more accurate. However, because *k*-gram prediction breaks down when the user is typing words that did not occur in the source corpus, *n*-gram predictions are often used as a fall-back. Character prediction generally relies only upon characters in the current word - although it is possible to use *n*-gram techniques across word boundaries, they are generally not very effective [Lesh et al, 1999a].

The Reactive Keyboard [Darragh et al, 1990] is a pioneering statistical letter prediction system that has influenced the design of Glyph. It was originally meant as an alternative interface for people with difficulties using the regular keyboard. However, the method has since been integrated into many other systems (e.g.: command completion in Unix). The system works by attempting to predict what the user intends to type next using the textual context seen so far. It uses the *n*-gram model trained using a large training set and subsequently refined with user input. The previous *n*-1 characters are used to guess the *n*th character.

Syntactic Word Prediction

A typical English sentence complies with the rules of English grammar. For example, in the sentence fragment “the quick brown fox”, we know that fox is a noun and that quick is an adjective. Part-of-speech (POS) tagging is defined as the process of assigning a part-of-speech or other lexical class marker to each word in a corpus [Jurafsky, 2000]. Syntactic word prediction systems use POS data to predict likely words or make informed decisions to refine candidate word lists. The richness of texting language

suggests that syntactic techniques will have limited success if applied to mobile text entry.

Statistical prediction systems may suggest words which are grammatically inappropriate in the specific context. This imposes an extra cognitive load on the user possibly even throwing the user's mind off the composition task at hand. Removing grammatically incorrect words from candidate lists have been found to improve the comfort of the user [Hunnicut, 1987].

Fazly [Fazly, 2002] has researched on the effect of syntax in word prediction. She reports that a strong unigram or bigram model is a good word predictor. However, there is an overlap in the information contained in a word bigram model and the information found in a POS tagged trigram model. There is also the issue of responsiveness of the system. The more information sources are consulted for prediction, the slower the system becomes. Thus, adding POS information to word bigrams and attempting to predict using both sources is not very effective.

Hybrid Techniques

Any word prediction method can be leveraged to generate character distributions by tabulating the probabilities of each character across each predicted word [Leshner et al, 1999a]. For example, if after typing "th" the predicted words and associated probabilities were (the:0.5, there:0.2, this:0.2, though:0.1), the predicted characters would be (e:0.7=0.5+0.2, i:0.2, o:0.1). Provided that the word prediction technique is more accurate than k-gram word completion, this approach will provide more accurate probability estimates than can k-gram or n-gram character prediction. A preliminary study using an ad hoc trigram (n=3) word prediction model to produce the character prediction list in a scanning system yielded an average keystroke savings of 2.4 percentage points [Leshner et al., 1998]. Recently, Li et al [Li et al, 2002] have reported impressive gains in Chinese script recognition using a combined approach using both character and word based bigrams.

2.3 Predictive Systems and the User

Early studies have shown that simple word prediction strategies reduced keystrokes by 23-58%, thus theoretically reducing fatigue and enhancing overall rates of communication. Along with decreased keystrokes, however, a number of cognitive and perceptual loads accompany the use of this type of tool [Koester et al, 1996]. Lists of words must be visually searched, the point of gaze must be shared between the keypad and the display, and a substantial amount of cognitive processing must be allocated to using and guiding the overall activity [Klund et al, 2001]. This is the classic *split-attention* problem in Human Computer Interaction (HCI). One conclusion, therefore, is that “even though there is a keystroke savings with the use of word prediction, there is not always an improvement in overall text generation or communication rate due to the costs of increased cognitive and perceptual loads” [Klund et al, 2001]. However, this conclusion is for word prediction systems and do not encompass character prediction systems.

While a significant body of literature discusses the keystroke savings and cognitive load issues associated with the use of word prediction software, there is no substantive work which examines the impact of such technology on the *quality* of one’s communication, or its ability to effectively represent the true cognitive and communicative capacities of the user. Fait et al [Fait et al, 2002] have shown that the use of word prediction technology, as compared to a more fluent style of communication,

1. Results in shorter responses,
2. Results in a less sophisticated writing style, and
3. Changes the qualitative nature of the communication.

Interestingly, text messaging on mobile devices can also be characterized by these very traits.

On a positive note, Magnuson et al [Magnuson et al, 2002] reports that long term use of word prediction applications can reduce cognitive load. Koester et al recognize that in the right situations, word prediction can enhance text entry. As a rule of thumb, word prediction is expected to increase text entry rate for users who type slower than 8wpm

(words per minute) and decrease the rate for users who type faster than 25wpm [Koester et al, 2002].

There has been no identified research on the effect of prediction specifically for mobile users although many of the issues identified for general predictive systems are applicable in this context. The benefits are expected to be more considering the limited input capabilities of the mobile device. However, the limited screen sizes of such devices do not lend themselves to the display of word completion lists. Statistical character prediction poses a lesser cognitive load and can be expected to increase text entry rate for ambiguous keypads. A critique of current systems based on these issues is provided in Chapter 3.

2.4 Models for Mobile Text Entry

The assessment and comparison of a new text entry method with current methods is a necessary part of the design process. The best way to do this is through an empirical evaluation. Unfortunately, such evaluations are time-consuming and complicated. Careful planning and execution is required when undertaking such an evaluation, as an abundance of confounding factors exists that could negatively affect its repeatability and validity. We focus on a discussion on the KSPC metric that can be used to compare current text entry systems. This metric is referred to in Chapter 3 where we critique current systems for mobile text entry and in Chapter 7 where we evaluate Glyph.

Key Strokes Per Character (KSPC) is the number of keystrokes required, on average, to generate a character of text for a given text entry technique in a given language [MacKenzie, 2002b]. KSPC is usually calculated using a linguistic model. One such model is Word Frequency in the language. For example, the top five words in the English language are 'the', 'of', 'and', 'to' and 'a'. For each word, we determine the keystrokes to enter the word in the interaction technique of interest. With this information, *KSPC* is computed as follows:

$$KSPC = \frac{\sum (K_w \times F_w)}{\sum (C_w \times F_w)} \quad (2)$$

where K_w is the number of keystrokes required to enter a word, C_w is the number of characters in the word, and F_w is the frequency of the word in the corpus.

The standard QWERTY keyboard obviously has a KSPC of 1 as it takes only 1 key to enter one character (lower case alphabets). The value increases if we consider all possible characters as certain characters require simultaneous depression of more than one key (Shift-2 for the '@' character). KSPC values for current text entry methods are discussed in Chapter 3.

Chapter 3

Evaluation of Current Mobile Text Entry Systems

Figure 3.1 shows current text entry systems based on Isokoski’s classification [Isokoski, 1999] and modified to the specific scope of the project, namely keypad based text entry for mobile phones. The diagram has also been updated to include recent systems although it is not a comprehensive list.

Figure 3.1. Classification of Text Entry Systems

Keyboard based systems utilize either a physical keyboard or a virtual keyboard displayed on a screen. Virtual keyboards have been compared by Kolsch et al [Kolsch et al, 2002]. While a discussion on virtual keyboard design is helpful to the project, we restrict our focus in this chapter to systems that are designed to work with the standard phone keypad. Key based systems are simple forms of text entry that do not incorporate any predictive features. Such systems include the Two Key method and MultiTap.

Intelligent systems are characterized by the use of techniques like word dictionaries, statistical linguistic information and Natural Language Processing to improve the effectiveness of text entry. Systems in this category include T9, HMS, EziText, LetterWise and Glyph.

Both MultiTap and the Two Press method are unambiguous. This means that given a particular sequence of key presses, the output is always the same. Given the ambiguous layout of the mobile phone keypad, this translates into a large number of extraneous keystrokes increasing the KSPC required. Intelligent methods aim to reduce KSPC at the possible cost of increasing computational requirements, cognitive load and resource consumption.

Several of the above-mentioned systems that have influenced the design of Glyph are critically analyzed in the following section. The text entry diagrams describing sample usage for each method are approximations made for clarity and are not a yardstick to compare the systems.

An analysis of current systems in the domain is also a critical element of framework design. Domain analysis is one of the first steps in framework development (Section 4.2). It involves studying as many systems as is practical from the target domain. Systems are analyzed to identify areas of specificity and generality. This is similar to the popular refactoring technique for classes albeit from a macroscopic perspective. Recurring patterns in the systems under study indicate possible candidates for refactoring into the framework. Areas that are specific to particular system are candidates for ‘pluggable’ implementations in the framework. Current text entry methods are analyzed with these goals in mind.

3.1 Key Based Systems

3.1.1 MultiTap

The MultiTap method is currently the main text input method for mobile phones. In this approach, the user presses each key one or more times to specify the input character. For example, the number key 2 is pressed once for the character 'A', twice for 'B', and three times for 'C'.

Figure 3.2. MultiTap

The MultiTap approach brings out the problem of *segmentation*. When a character is placed in the same key as the previously entered character (e.g., the word 'on' requires the user to press 'o' followed by 'n' both of which are found on the '6' key), the system must determine whether the new key press still "belongs to" the previous character or represents a new character. Therefore, a mechanism is required to specify the start of a new character. There are three main solutions to this problem.

Timeout: With this approach, a timer is started each time a key is pressed. If the user presses the key again within the timer period, the display cycles to the next character that is mapped to the particular key. The disadvantage is that, to enter two successive characters that happen to be mapped to the same key, the user must first wait for the whole timer period to expire. Most phones have a timeout, typically between 1 and 2 seconds.

'NEXT' Button: In practice, users have to wait for the timer to expire frequently (e.g. 'on' is a common word). Rather than waiting for a timeout, a special button can be used to confirm the entry of a character and to move on to the next character. Using the example of entering "no", the user would again press the six key twice to enter an 'n'. They would

then press the 'NEXT' button to confirm the entry of the 'n'. The 'o' could then be entered with three presses of the six key.

Hold Down: This method is similar to the *timeout* method with the difference being that the system automatically cycles through the different characters mapped to a particular key when that key is held down. Segmentation occurs automatically when the key is released

Some phone models use a combination of solutions. For example, Nokia phones include both a 1.5-second timeout and the provision for a timeout kill using the arrow keys. The user can decide which strategy to use.

As its name implies, the MultiTap method has a KSPC greater than one, unless the 'Hold Down' strategy is used, in which case its KSPC will be close to 1.0, though the KSPC value would be a poor measure of the efficiency of this segmentation strategy. MacKenzie [MacKenzie, 2002] calculated the KSPC for the MultiTap with next method to be 2.0342. This means that on an average, two key presses are required to input a single character of text.

MultiTap typically requires more key presses than most other methods. However, there is no ambiguity and all text typed by the user is immediately *committed*. This lack of ambiguity makes MultiTap highly useful as a fall back method for advanced predictive systems.

3.1.2 Two Key Method

In the Two Key method, the user presses two keys successively to specify a character. The first key press, as in the MultiTap method, selects the "group" of characters (e.g., key 5 for 'JKL'). The second press is for disambiguation: one of the number keys, 1, 2, 3, or 4, is pressed to specify the position of the character within the group. For example to enter the character 'K', the user presses 5-2 ('K' is second character in 'JKL').

This technique means that every letter takes exactly two presses to enter, unlike MultiTap where the number of presses ranges from one to three (or more for punctuation). Two Key, however, is not suited for entering punctuation or special characters, as the user needs to be able to see all the letters mapped to each key in order to determine the

position of the key they desire. The Two Key method is not in common use for entering Roman characters. However, in Japan, a similar method (often called the "pager" input method) is very common for entering Katakana characters.

3.2 Intelligent Systems

3.2.1 LetterWise

LetterWise [Mackenzie et al, 2001] by Eatoni uses character level prefix information for disambiguation. It works with a stored database of probabilities of prefixes. A prefix is the letters preceding the current keystroke. For example, if the user presses 3 with prefix 'th', the system suggest 'the' because 'the' is statistically more common in English than the alternative candidates, 'thd' and 'thf'.

In the event that the choice of letter offered by the system is incorrect (in the sense that the user intended to input another word and hence another character), the user presses the NEXT key repeatedly until the correct character is displayed.

LetterWise does not use a dictionary of words. Instead, an a priori analysis of a dictionary is used to obtain probability information for character sequences. A selected subset of these prefixes is then used in the production system.

Figure 3.3. LetterWise

A second method developed by Eatoni is WordWise¹ which takes a rather different approach. It uses a system where the letters 'c', 'e', 'h', 'l', 'n', 's', 't', and 'y' are entered unambiguously using an auxiliary key. For example, the 5-key is mapped to the letters 'j', 'k' and 'l'. Pressing 5 by itself means the user wishes to enter a 'j' or a 'k', while holding

¹ <http://www.eatoni.com/wordwise/>

the auxiliary key and pressing 5 will always enter an 'l'. This allows words like "yes" to be entered without the system having to make any predictions. It also reduces the number of possible meanings of a group of key presses as the letters that are entered unambiguously narrow the possibilities. Entire words (such as "the" and "then") can also be entered unambiguously, which means they do not need to be stored in the dictionary. However, there is still the issue of unknown words, which must be entered using MultiTap, as with T9.

3.2.2 T9

The T9 input method [Tegic, 2003] uses a dictionary as the basis for disambiguation. The method is based on the same key layout as MultiTap, but each key is pressed only once. For example, to enter "the", the user enters the key sequence 8-4-3-0. The 0-key, for SPACE, delimits words and terminates disambiguation of the preceding keys. T9 compares the word possibilities to its linguistic database to predict the intended word.

Figure 3.4. T9

Linguistic disambiguation is not perfect, since multiple words may have the same key sequence. While T9 reduces character level ambiguity, it introduces word level ambiguity. In these cases, T9 gives the most common word as a default. To select an alternate word, the user presses a special NEXT function key. For example, the key sequence 6-6 gives "on" by default. If another word was intended, the user presses NEXT to view the next possible word. In this case, "no" appears. If there are more alternatives, NEXT is pressed repeatedly until the intended word appears. Pressing 0 accepts the word and inserts a SPACE character. During this phase, the user is in *composition* mode which is reflected in the display using visual cues like text color inversion. Pressing 0 accepts the word and inserts a SPACE character. The text is now said to be *committed*.

T9 disambiguation has been found to work quite well in practice [Silfverberg, 2000]. In a sample of the 9025 most common words in English produced from the British National Corpus, the user must press NEXT only after about 3% of the words. Naturally, the whole vocabulary is larger than 9025 words, so this estimate may be optimistic.

A major problem with T9 is that of *word instability* [James et al, 2001]; the word display continually changes as the user enters each additional character to the word. In addition, T9 “blows up” in the face of user errors; users typically have to clear the current word and start again from scratch.

T9 supports personalization by allowing users to add custom words to the dictionary. However, this is a comparatively complex task requiring the user to halt his message composition, switch to an alternate text entry method (MultiTap) and enter the new word. Several service providers also allow users to download custom dictionaries (e.g.: different languages) to their phones. Another commercial implementation of a dictionary based system is Motorola’s iTap².

Most major mobile phone manufacturers have licensed the T9 input method, and it is the de-facto method in most commercial products.

Dunlop and Crossan [Dunlop et al, 2000] have independently developed a similar system that uses a large dictionary of words in the language of usage of the phone and, furthermore, all morphological variants of these words which are used in the language together with information on how often each variant is used in the language. They propose an extension of this model to give the user the ability to carry out automatic word completion. When a user starts a word, the most likely word with that start could be proposed as a suggested word for auto completion thus further accelerating the input process.

The KSPC of T9 has been calculated at 1.0072 [MacKenzie, 2002]. However, this calculation assumes that all words are found in the dictionary. In addition, it does not take into account key strokes required to select candidate predictions or to correct errors. Due

² http://www.motorola.com/lexicus/html/itap_FAQ.html

to the problems plaguing the T9 system as mentioned in the previous paragraph, the number of such key strokes is likely to be a sizeable amount.

3.2.3 EziText

EziText [ZiCorp, 2003] is similar to T9 in that it uses a dictionary based disambiguation system. However, EziText improves on T9 by providing candidates for completing the current word that the user is entering. Users do not need to tap multiple times to reach letters because EziText predict words before all letters are pressed. Possible completions for the user's input are displayed at the bottom of the screen and the best completion is displayed inline ahead of the *composing* fragment of text (black). The predicted text, with its visual cues is called *lookahead* text (light gray).

Figure 3.5. EziText

The system places special emphasis on learning and personalization. Usage patterns are learned from several sources within the user interface system. Phonebook entries are provided as candidates for completion during text messaging. It also stores information about frequently used words to provide better candidates in the event of a disambiguation conflict. However, an additional requirement introduced by word completion systems is the need for screen space to display possible completion candidates.

KSPC for EziText is not provided but can be expected to be around 0.7, the average figure for word prediction systems.

3.2.4 HMS

The HMS system [Hasselgren et al, 2003] is a relatively new method still in the research phase. The system augments the standard word dictionary with word bigram (two consecutive words) statistics, to give a better text prediction. The list of bigrams is stored in memory together with their frequency of occurrence and it is accessed simultaneously with the character input. Given a previously written word, the most probable subsequent words are extracted from the bigram list.

The HMS system has been trained using a corpus of newspaper text; the lack of a suitable text messaging corpus has been cited as the reason. The system also has relatively high memory requirements due to the need to store word bigram information.

3.3 Other Systems

The systems analyzed so far have all been software based. A large category of text entry systems exist that use innovative hardware designs to improve text entry rate. An extensive review was performed during the initial stages of the domain analysis. A brief overview of selected designs is presented below.

Figure 3.6. The FastTap keypad

In the FastTap³ keypad, the letters sit on raised bumps between the rounded number and punctuation keys. Although the letter bumps are individually small because they sit between the larger number keys, there is enough room to ensure that pressing one does not accidentally include another.

Figure 3.7. The Q12 keypad

³ <http://www.digitwireless.com/overview/overview.html>

Q12⁴ is a text input method for mobile phones that enables entering of any letter with one press of a finger. It is achieved by changing the shape of buttons of the traditional phone keypad.

Figure 3.8. ThumbScript

ThumbScript⁵ uses a visual connect-the-dot strategy on a 9 key keypad for text entry. Letters are entered by pressing once where the stroke of a letter begins and once where it ends.

⁴ <http://www.softava.com/q12/concept.htm>

⁵ <http://www.thumbscript.com>

Chapter 4

The Prototyping Framework: Design and Implementation

The effectiveness of many computing systems can be proven with objective tests and quantifiable results. Text entry methods however interface directly and intimately with the end user. Subjective satisfaction is also important because if users do not like the newer technologies of text entry, they are less likely to buy a phone which offers that technology. Users like to stick with things that are more familiar to them, that are available on older phones and that they already know how to use. Introducing new methods of data entry often causes apprehension and mistrust before the users actually use the technology. An empirical evaluation with users is therefore, paramount to the viability of new methods of text data entry [Friedman et al, 2001]. Designing new text entry methods for computing systems is a labor intensive process. It is also expensive, since a working prototype must be built, and then tested with real users [Silfverberg et al, 2000].

Developing a high-fidelity (typically, highly interactive) prototype for mobile applications is not an easy task. Often, the text entry researcher may need the help of a mobile application developer in converting new text entry algorithms into a working prototype that is sufficiently complete for user testing. The problem is compounded if the researcher wishes to test custom key layouts or opts for an entirely new device design. In the past, applications for mobile devices were designed to work within the constraints of the target device. There is an increasing trend, primarily in the mobile phone market, to tailor phone designs for key applications like SMS (The Nokia 6800¹ allows the user to flip open the phone cover to reveal a full size keyboard to ease text entry).

While rapid prototyping and presentation tools like Macromedia Director are useful for explaining a system to users, useful testing requires a working prototype that is a close

¹ <http://www.nokia.com/nokia/0,8764,4486,00.html>

match to the final system. In an ideal world, a prototype would be completely separate from the final product. However in the fast paced environment that is software development, it is an economic necessity to reuse components of the software prototype [Purtilo, 1991].

The current crop of phone simulators serve only as an application testing aid and do not address the requirements for usability testing or quick prototyping. They are also highly manufacturer and device specific. Prototyping a new text entry method by developing it on an actual phone or phone simulator is akin to premature optimization.

Recent years have seen many text entry methods being designed. However, these projects typically expend substantial effort in developing their own prototypes for presentation and user testing. A better technique would be to develop for a well-designed generic framework (or even interface existing code to the framework) to rapidly prepare a realistic prototype which can then be tested internally or with target users using a variety of usability testing methods.

This chapter describes the conceptualization, design and development of the Prototyping Framework (henceforth referred to as the *framework*). The primary goal of the framework is to help text entry researchers produce high-fidelity working prototypes of their concept. It aims to ease development effort and maximize reusability of code (Section 4.6). The framework is adaptable to handle innovative device designs and sufficiently flexible in its design to enable a developer to customize interactions specifically for his project.

4.1 Object-Oriented Application Frameworks

Reuse of software has been a goal in software engineering for almost as long as the existence of the field. Several research efforts have aimed at providing reuse [Mattson, 1996]. During the 1970s, the basics of module-based programming were defined and software engineers understood that modules could be used as reusable components in new systems. Modules, however, only provided ‘as-is’ reuse and adaptation of modules had to be done either by editing the code or by importing the component and changing those aspects unsuitable for the system at hand. During the 1980s, the object-oriented languages increased in popularity, since their proponents claimed increased reuse of

object-oriented code through inheritance. Inheritance, different from importing or wrapping, provides a much more powerful means for adapting code.

However, all these efforts only provided reuse at the level of individual, often small-scale, components that could be used as the building blocks of new applications. The much harder problem of reuse at the level of large components that may make up the larger part of a system, and of which many aspects can be adapted, was not addressed by the object-oriented paradigm in itself. This understanding lead to the development of object-oriented frameworks.

Early examples of the framework concept can be found in literature that has its origins in the Smalltalk environment (One of the first OO languages) [Goldberg et al, 1983]. The Smalltalk user interface framework, Model-View-Controller (MVC), was perhaps the first widely used framework. Since its conception, it has served as the core architecture for numerous applications and frameworks.

Figure 4.1. The decomposition of an application

Numerous definitions of the term “Framework” have been proposed. Booch offers a generic definition [Booch, 1996]: “Framework : A collection of classes that provide a set

of services for a particular domain; a framework thus exports a number of individual classes and mechanisms that clients can use or adapt.”

An application can be composed of multiple frameworks as shown in Figure 4.1. These in turn can be broken down into design patterns or even classes.

4.2. Framework Development Approaches

Designing a single class is an order of magnitude harder than designing a single function. Similarly, designing a framework is much harder than designing a class library or an application because the framework developer has to think ahead and anticipate ways in which a user will use or abuse the framework. This difficulty has been identified in almost all major works on the subject.

Standard OO design techniques must be modified to support the framework development process. Various development approaches and guidelines have been proposed to ease the effort required [Fayad et al, 1999]. A combined approach is shown in Figure 4.2.

The development process based on this approach is outlined below:

Figure 4.2. The iterative framework development process

4.2.1 Domain Analysis

Frameworks are targeted at specific domains and therefore, the development process starts with domain analysis. Applications in the domain must be analyzed and common abstractions should be extracted. The scope and specific areas that the framework targets should be decided at this stage. The bottom-up approach is a good method to identify abstractions. Data structures and algorithms should be analyzed between applications. As in the case of this project, if the developer is not familiar with the application domain, it is suggested that three different applications be developed targeting the domain. The domain analysis performed for developing the prototyping framework was discussed in Chapter 3.

4.2.2 Design

Once the framework goals are finalized and abstractions organized, the framework is ready to be designed. Popular approaches include Hook-Template pairs [Froehlich et al, 1997], Design by Contract [Horstmann, 1997] and application of Design Patterns [Gamma et al, 1994]. A very critical decision at this stage is the design of client-framework interaction.

4.2.3 Implementation and Testing

The core abstract classes are implemented during this stage. Additional concrete classes can also be written. One or possibly a few applications are developed based on the framework. This is the testing activity of the framework. Problems when using the framework in the development of the applications are captured and solved in the next version of the framework.

4.2.4 Refinement

Building a framework is an iterative process. The developer updates the framework design using insights gained by developing applications using the framework. The problem domain might also need to be re-analyzed.

4.3 Framework Design Elements

When developing a framework, rather than an ordinary object-oriented application, some program constructs, or “design elements”, are very important. Such design elements include abstract classes, object-oriented design patterns, dynamic binding and contracts.

The lack of suitable support for such constructs in existing object-oriented methods has reported by Bosch [Bosch et al, 1999]. In addition to the elements mentioned here, standard object-oriented features like function overloading and templates are also useful.

4.3.1 Dynamic Binding

Typical function calls are completely determined at compile time. This is called static binding. Dynamic binding occurs during run time when the actual method that is executed depends on the type of pointer invoking the method. This is perhaps one of the most important design elements for framework design. If we compare a framework with a conventional library, we can appreciate the importance of dynamic binding. When a conventional library is used by an application, calls are made from the application to the library only. In an object-oriented framework, however, calls typically can go also in the opposite direction, as shown in Figure 4.3. This inversion of control is made possible by dynamic binding.

Figure 4.3. Inversion of control in a Framework

4.3.2 Interfaces

An interface specifies the services provided by a class. It is the list of all public methods exported by a class. Java directly supports the concept of interfaces. In C++, abstract classes are used for creating interfaces. Interfaces are the base concept for the design by contract method.

4.3.3 Abstract classes

Concrete classes are classes that can be instantiated (that is, of which objects can be created). Most classes are typically concrete. However, there are some classes that cannot be instantiated and are present only to abstract a concept. Such classes are called abstract classes. Horstmann refers to them as abstract base classes because they are typically used

as a base class for derivations and as a placeholder for virtual functions that need to be implemented by subclasses [Horstmann, 1997]. Frameworks use abstract classes to enforce consistency in interfaces.

4.3.4 Hooks and Hot Spots

Application specific extensions are connected to the framework through the concept of hooks. Hooks are locations in the framework that can be extended in some way to provide application specific functionality.

Hot Spots are general areas of variability in the framework where placing hooks might help. They correspond to the volatile aspects discovered in the domain analysis.

4.3.5 Design Patterns

The concept of patterns has its origin in the work by Christopher Alexander, an architect who developed the idea of a pattern language to enable people to design their own homes and communities. The idea of patterns has been adopted by software engineers in object-orientation, who transformed the ideas of Alexander to the area of software development. The main approach taken by the software engineers is to develop patterns that are solutions to small-scale design problems. Perhaps the most gratifying aspect about patterns is that it is a concerted effort by expert designers to organize and publish verified designs arising from their own experience.

The pioneering work on design patterns defines them as:” *Descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context*” [Gamma et al, 1994].

4.4 Framework Design

The core classes of the framework are outlined in Figure 4.4. The *Facade* design pattern is used extensively in the framework to hide complexity. The functionality provided by the framework is accessed through a set of `Manager` classes. For example, the `KPManager` class hides the intricacies of keyboard management behind easy to use methods. The managerial classes also use the *Singleton* pattern to ensure that only one instance of the class can be instantiated in the framework scope.

Figure 4.4. Core Classes

4.4.1 Core Classes

DeviceManager: This class is responsible for the entire visible UI presented by the framework. It uses instances of `KPManager`, `ScreenManager` and `TextManager` to handle the dynamic aspects of the framework. These aspects include receiving, filtering and forwarding user input, managing the screen display and process text.

Project Manager: Every application that makes use of the framework is required to create a project file describing the application specific configuration of the framework. Typical configuration data includes information about custom classes that are designed to

replace certain aspects of the framework and a layout file that defines the visual look of the device.

KPManager: This class abstracts low level details of keyboard handling and key binding. The physical keyboard is remapped using the configuration data supplied by the application. The framework also includes printable templates for labeling remapped keyboards. Raw keyboard input is filtered and processed into high level framework objects (eg: KPButton) before being passed on to the framework for event handling. It also handles the requirement for button timeouts as identified in the case study.

ScreenManager: The ScreenManager handles text display using the Screen and ScreenDocument classes. This design is based on the Model-View-Controller architecture.

4.4.2 Text Processing

At the heart of the framework is the text processing system. Text processing is the gist of any modern text entry method. It is therefore the most variable aspect across applications as identified in the case studies. The TextManager class provides the facade to access this system. A hook is provided in the form of the TextProcessor class. Client applications using the framework are expected to write their custom implementations to accommodate their specific text processing algorithms. If a custom implementation is specified by the project, the TextManager routes all text processing events through the custom TextProcessor.

The concept of *committed*, *composed* and *lookahead* text fragments were introduced in the case study. These are discussed in further detail in this section. In the MultiTap system, lack of ambiguity means that all text entered by the user is directly committed to the model. In the T9 system, the disambiguation algorithm does not always provide the word intended by the user. The user is in composing mode until he confirms a chunk of text in which case it is committed. Many devices also let users type text in Chinese, Japanese, or Korean, languages that use thousands of different characters, on a regular-sized keyboard. The text is typed in a form that can be handled by regular-sized keyboards, for example, in a romanized form, and then converted into the form that's really intended. Typically a sequence of several characters needs to be typed and then

converted in one chunk, and conversion may have to be retried because there may be several possible translations.

Figure 4.5. Text Processing

While this "composition" process is going on, the text logically belongs to the text processing model, but still needs to be displayed to the user. Modern conventions dictate that the text be displayed in the context of the document that it will eventually belong to, albeit in a style that indicates that the text still needs to be converted or confirmed by the input method. This is called *on-the-spot editing*. Some systems provide a separate window to display the text. Such systems are said to use *root-window editing*.

State of the art methods like EziText add another type of fragment, the *lookahead*. Using the current context, they make predictions as to what the user intends to type in the future. This fragment also requires unique visual cues to distinguish itself clearly to the user.

Extrapolating from the previous discussion, we can argue that future methods might require additional fragment types or might drop current types. This is clearly yet another area in the framework that should be opened up for application specific customization. The framework handles these aspects using the `TextFragment` and `TextContext`

classes. A `TextFragment` encapsulates a string of text. It is also associated with a visual style. A `TextContext` is an ordered list of named `TextFragments`. An application, on startup, registers its fragment types and associated visual styles with the `TextManager`. This information is stored in a `TextContext` which is then passed to the text processing engine each time a text event is encountered. The `TextContext` holds all relevant information about text entered by the user so far.

4.4.3 XML driven configuration

The current version of the framework uses two XML configuration files.

`Project.prj` : This file contains information about the a project. This includes type information and location of custom classes used and the path to the layout file.

```
<layout>

  <!-- Device Specifications -->
  <device img="images/Simulator.jpg">
    <!-- Dimensions of the device, all children are positioned relative to this -->
    <dimensions x="0" y="0" width="198" height="405"/>

    <!-- Button Layout -->
    <!-- [offsetXY] is used to offset the entire button layout group -->
    <buttonLayout offsetX="0" offsetY="0">
      <!-- Buttons 0-9 -->
      <button name="1" id="1" keystroke="NUM_LOCK" charlist="1" img="images/KPB_1.jpg"
        <dimensions x="37" y="259" width="40" height="22"/>
      </button>
      <button name="2" id="2" keystroke="DIVIDE" charlist="abc2" img="images/KPB_2.jpg"
        <dimensions x="81" y="263" width="40" height="22"/>
      </button>
      <button name="3" id="3" keystroke="MULTIPLY" charlist="def3" img="images/KPB_3.jpg"
        <dimensions x="126" y="259" width="40" height="22"/>
      </button>
      <button name="4" id="4" keystroke="NUMPAD7" charlist="ghi4" img="images/KPB_4.jpg"
        <dimensions x="37" y="283" width="40" height="22"/>
      </button>
      <button name="5" id="5" keystroke="NUMPAD8" charlist="jkl5" img="images/KPB_5.jpg"
        <dimensions x="81" y="286" width="40" height="22"/>
      </button>
      <button name="6" id="6" keystroke="NUMPAD9" charlist="mno6" img="images/KPB_6.jpg"
        <dimensions x="126" y="283" width="40" height="22"/>
    </buttonLayout>
  </device>
</layout>
```

Figure 4.6. Layout.xml

`Layout.xml` : Each project requires a layout file that describes the layout of the mobile device in terms of dimensional information and custom attributes if any. A sample `Layout.xml` is shown in Figure 4.6. This information is used by the `DeviceManager` to render the device UI. An important feature of the framework is its

ability to support custom attributes for the input buttons defined in the layout. The importance of this feature is discussed in the implementation section.

Figure 4.7. Startup Sequence

4.4.4 Visual Design Considerations

Usability testing is expected to be a major application for the framework. Therefore, special considerations have been made in the visual design of the framework. The framework renders to a full screen window (with no embellishments) and the device UI is embedded in the center. This is to avoid distracting visual clutter and to help the user to focus on the task at hand. The buttons also support visual feedback on key events. This is a debatable feature and one that requires further investigation.

4.5 Application Development using the Framework

A good method for testing the applicability of a framework to its task domain is to develop applications using it. With this goal in mind, two text entry methods, MultiTap and T9, were developed and integrated into the framework. This section outlines the key steps a developer has to follow to configure and use the framework with his application.

1. Product Design

The developer has a choice of using one of the standard device designs provided by the framework. However, the framework is in its element when handling innovative product design (see Figure 4.8). The static image of the product is segmented into logical sections like buttons, screen etc.

Figure 4.8. Building a Custom Device Interface

2. Layout and Attributes

The layout of the device’s component elements are defined using a layout.xml file. At this point, any custom attributes required by the text processing engine are also added. The T9 implementation uses a modified tree data structure to store the word dictionary. Searching this data structure in an efficient manner requires the application to obtain a special code from the currently pressed button. To configure the framework to support this requirement, a custom button class is written with an accessor method for this code. The button element in the layout.xml is given the code attribute. The DeviceManager, upon detecting the new attribute, uses reflection to introspect the custom button class to detect the correct accessor method thus making the button code available to the text processing engine. The T9 implementation is shown in Figure 4.9.

3. Code Integration

A custom TextProcessor class is written and linked to the framework using the project file. This class registers with the framework, the fragment types it intends to use along with their associated visual styles.

The framework can now be launched and once pointed to the project file, the prototype is configured, rendered and ready for testing.

Figure 4.9. A T9 implementation using the Framework

4.6 Evaluating the effectiveness of the Framework

An analysis of current research projects in mobile text entry show a wasteful trend. Most such projects have innovative ideas for improving text entry. However, they make minimal changes to the user interaction strategy and end up developing applications to test their concept. The innovation typically lies in the language model and backend algorithms. Understanding this environment, we can see that there is a need for a framework to conceptualize and prototype the text entry system. This is to allow researchers to focus on their core contribution without spending costly time and effort on building systems to prototype their idea. We have not been able to identify any similar frameworks in the field and hence a comparative evaluation is not possible. It is hypothesized that the lack of similar framework is due to individual projects not being able to spend substantial effort on generalizing and implementing a reusable framework when a specific application would serve their immediate needs.

It is a well-known fact that framework development is an iterative process. Traditional software development also requires iterations but iterations are more important and explicit when designing frameworks. The underlying reason, according to Bosch et al. is that frameworks primarily deal with abstractions, and that abstractions are very difficult to evaluate [Bosch et al, 1999]. Therefore, the abstractions need to be made concrete in the form of test applications before they can be evaluated. Due to the limited time frame allocated to this part of the project, application development using the framework was limited to three similar applications, namely implementations of MultiTap, T9 and Glyph (by similarity, we refer to the method of integration with the framework and not the actual applications themselves, which are quite different.). While this does not afford enough information to evaluate the design thoroughly, a basic evaluation is still possible.

Froehlich et al. has identified several desirable features of a framework [Froehlich et al, 1997]. The framework is evaluated on the basis of these and other relevant criteria.

Degree of Reuse

Figure 4.10 shows the minimum amount for code required to implement a MultiTap system using the prototyping framework. The code saving and reduction in programming effort are obvious as the framework code performs the majority of the extraneous work

required to configure and display the device interface and interact with the user. The programmer is free to focus solely on the task of developing the text entry algorithm.

The framework is aimed at rapid prototyping and user testing. Once the efficacy of a text entry method has been verified through the prototype, the text entry algorithms must be, in all likelihood, rewritten and optimized for the actual mobile device. However, a fair amount of conceptual reuse can be expected even in this process.

Ease of Use

Ease of use refers to an application developer's ability to use the framework. The programming model adopted for the framework is intuitive providing the developer with high level abstractions for textual context and user input. For example, the `TextContext` class logically abstracts the data displayed in the interface into clearly demarcated and easily modifiable `TextFragments`.

Extensibility and Flexibility

A framework is extensible if new components can be added with ease. Flexibility is the ability to use the framework in many contexts. Core classes of the framework can be replaced with application specific polymorphic variants to add additional functionality. However, in its current stage of development, swapping the core classes require the client programmer to be familiar with the design of the backend interactions.

There is also a heavy dependence on the Java Swing API implying that developers are limited to the generic functionality provided by the Swing classes. This dependence can be minimized by developing a set of classes to manage the visual look of `TextFragments`.

The framework is flexible enough to be integrated with most text entry systems. However, truly innovative systems which require special and major changes to the interaction model will require the developer to extend and customize the framework. The framework is still in its infancy and additional applications need to be developed using it to iron out rigidities and add features that are required.

Consistency

Consistency in coding style, interfaces and naming conventions are strictly enforced in the framework. Consistency reduces the time taken by developers to understand the framework and helps reduce errors in its use.

```

public void initialize( TextManager textMngr )
{
    // Add fragment types
    SimpleAttributeSet committedStyle = new SimpleAttributeSet();
    StyleConstants.setForeground( committedStyle, Color.BLACK );

    SimpleAttributeSet composingStyle = new SimpleAttributeSet();
    StyleConstants.setForeground( composingStyle, Color.WHITE );
    StyleConstants.setBackground( composingStyle, Color.BLACK );

    textMngr.addFragmentType( 0, "committed", committedStyle );
    textMngr.addFragmentType( 1, "composing", composingStyle );
}

public TextContext processText( TextContext inputContext, KPButton button )
{
    if( button.isTimerReset() )
    {
        String prevComposing = inputContext.get( "composing" ).getFragment();
        inputContext.get( "committed" ).appendFragment( prevComposing );

        inputContext.get( "composing" ).setFragment( button.getCurrentChar() );
    }
    else
    {
        inputContext.get( "composing" ).setFragment( button.getCurrentChar() );
    }

    return null;
}

```

Figure 4.10. Minimum Code to Implement MultiTap using the Framework

Cohesion and Coupling

A class is cohesive if it has a well defined purpose and all its members are implemented in such a way as to achieve this purpose. Classes in the prototyping framework exhibit a high degree of cohesion. An example is the `KPButton` class whose sole purpose is to abstract a keypad button along with its associated functionality. Its methods, like `startTimer()`, work towards achieving that purpose.

Coupling refers to the linkages between classes and should be minimized. In the case of framework, coupling is unavoidable in certain cases like in design of the core classes and within individual packages. Bi-directional associations between classes have been avoided where possible.

Chapter 5

The Design of Glyph: An Efficient and Adaptive Text Entry System

5.1 Overview

The design of Glyph builds on research dating back to the 70's adapted to the requirements of the modern mobile environment. This chapter describes the design goals for Glyph. It explains various design issues and their solutions, the tradeoffs made and their justifications.

Content analysis forms a significant part of the design process as it provides valuable insights into the nature of text messaging. The chapter begins with an overview of the nature of mobile text messaging and the data sources used in building Glyph.

5.2 The Nature of Mobile Text Messaging

Text messaging is arguably the most demanding text entry task performed on a mobile phone. Each short message is up to 160 characters in length when Latin alphabets are used and 70 characters in length when non-Latin alphabets such as Arabic and Chinese are used. According to GSM Association figures, about 366 billion SMS messages were sent globally across GSM networks in 2002.

The majority of active text users are between 14 years of age and 40 [JustText, 2003]. One out of 3 people in the world today is a teenager. US teens comprise 13% of the population, the largest generation in history [US Census, 2000]. With greater access to technology than any before, today's youth are coming of age with the rapid growth of the Internet and global adoption of mobile phones and other wireless devices. Social connection and communication have always been of fundamental importance to teenagers. What's new is the panoply of media options available to this "instant messenger (IM) generation" [Lenhart et al, 2001], whose mobile phone ownership is predicted to soon reach 85% by age 18 [Grinter et al, 2003]. As the younger generations

(the biggest users of text messaging) grow up, they will take their texting skills with them. They will continue to educate the older generations and will also pass on their skills to the new generation.

As the world's mobile networks work their way to high-speed, always-on "3G" wireless connectivity, experts are pointing to mobile instant messaging (IM) as one of the emerging service's potential killer apps [Saunders, 2003]. Fortunately, unlike conditions in desktop IM, a number of major players are now actively pursuing deals to promote interoperability. Wireless carriers are increasingly interested in rolling out full-fledged mobile IM in addition to the more simple SMS, which lacks the presence and availability information central to IM Buddy Lists and similar features of other networks. Like SMS, encouraging users to adopt mobile IM is important for increasing carriers' revenues, since most users of such 2.5G or 3G services will pay per unit of data transferred, rather than per-message.

Simple text messaging on mobile phones already poses significant challenges in the design of text entry methods. The arrival of mobile IM exacerbates the problem by elevating the importance of text entry rate. One way in which users have tried to overcome this hurdle is by creating an evolving language using shortened words. Four methods emerged for generating these shortened words: using traditional (known) or ad-hoc abbreviations; dropping a single letter; using letters, symbols or numbers to make an appropriate sound; and using standard or ad-hoc acronyms. The two most common methods for shortening words were to use letters, symbols or numbers to make an appropriate sound (37%) and the use of abbreviations (47%) [Grinter et al, 2003].

Studies of Finnish and German text messages show that teens use English language shortened forms. In addition to using computer/Internet terminology, teens in these countries also use shortened words like C for see. Instead of using long challenging phrases offered by dictionaries, the teenagers recorded shortening simple words such as tomorrow and weekend, which often appeared in messages discussing plans. Other commonly shortened words included school, football, Internet, lessons, and homework.

Table 5.1. Type of shortened word used in messages

Type of Short Form	Percentage
Abbreviations: Ad-hoc (spose for suppose)	18%
Abbreviations: Known (mins for minutes)	14%
Dropping Single Letter (ritten for written)	15%
Sounds: Letters (fone for phone)	25%
Sounds: Symbols (th@s for that's)	3%
Sounds: Numbers (gr8 for great)	9%
Acronyms: Separate Words (PWB for please write back)	4%
Acronyms: Single Word (w for with)	1%
Acronyms Compound Word (gf for girlfriend)	1%
Hybrid: Using two or more of the above (b4 for before is a letter drop & number sound and ThanQ is a letter drop & letter sound)	5%
Foreign Short Forms (bs for pesos)	4%
Foreign Letters (ü for -))	<1%

In summary, the teenagers used shortened words that were concerned with practical matters such as meeting up at the bus stop rather than the more complex expressions offered by SMS dictionaries. As Crystal [Crystal, 2001] observes, many words in “new word” dictionaries do not typically make it into everyday language. Abbreviations and acronyms that serve everyday needs (not just useful words, but also short forms that disambiguate or add richness to the medium such as ü) appear to be those appearing in text messages.

The research highlighted above has significant implications in the design of mobile text entry methods. It suggests that dictionary based systems that do not adapt to the user’s language are bound to be shunned by users. Such systems force the user into composing text in valid English words without the richness of their own personal lingo. We hypothesize that dictionary based systems taken alone cannot cope with the requirements of mobile users. Messaging styles are unique even within relatively small communities. The Straits Times [Straits Times, 2001] reports that the predominant language style used by Singaporeans for text messaging is Singlish (Singapore Colloquial English).

Current systems require substantial effort on the part of the user to add personal words to their dictionary (Chapter 3). Bundling phones with custom dictionaries tailored to a particular user demographic is only a partial solution as every user is unique in their messaging style; some more so than others.

5.3 Data sources for the construction and testing of language models

5.3.1 Overview

The corpus in Computational Linguistics refers to a large and structured set of texts usually electronically stored and processed. For example, the British National Corpus (BNC) is a 100 million word collection of samples of written and spoken language from a wide range of sources, designed to represent a wide cross-section of current British English, both spoken and written [Clear, 1993].

SMS is a relatively new communication medium. It has not been subjected to the kind of analysis afforded to email [Petrie, 1999], IRC [Werry, 1996] and other new age media. The lack of a high-quality SMS corpus has often been cited by many researchers as an obstacle in developing and testing innovative text entry methods for mobile devices.

The first half of the project used data from the BNC corpus, phrase sets and the raw SMS corpus. The National University of Singapore (NUS) recently released an annotated SMS corpus that is used as the primary data source for Glyph. Brief descriptions of these corpora are provided in the following sections.

Table 5.2. Corpora used by the project

Corpus Name	Short Name	No. of words	No. of sentences
NUS SMS Corpus	SMS1	124,995	~10,000
British National Corpus	BNC	>100M	N.A
Bergen Corpus of London Teenage Language	COLT	>500,0000	N.A
Phrase Sets for Text Entry Research	PSTER	2,713	500
Web SMS Corpus	SMS2	3,884	212

5.3.2 NUS SMS Corpus

The NUS SMS Corpus [How, 2004] is a corpus of SMS messages collected for research at the Department of Computer Science at the National University of Singapore. The corpus consists of about 10,000 SMS messages, largely originating from Singaporeans

and mostly from students attending the University. These messages were collected from volunteers who were made aware that their contributions were going to be made publicly available.

The bulk of the corpus (60%) is formed by messages collected on a regular basis from several specific phone users aged between 18 and 22 years. The remaining content of the corpus is composed of web based submissions.

According to the researchers, a total number of 3,348 SMS messages were collected from 146 people using the website. This large number of users contributing to the website shows that messages were collected from a wide range of users and this gives sufficient breadth to the corpus of messages collected.

As is the current trend, the corpus is provided as an XML document along with its associated schema. The cleaned up corpus contains 8298 tokens.

5.3.3 British National Corpus

The British National Corpus (BNC) is a very large (over 100 million words) corpus of modern English, both spoken and written [Clear, 1993]. The Corpus is designed to represent as wide a range of modern British English as possible. The written part (90%) includes, for example, extracts from regional and national newspapers, specialist periodicals and journals for all ages and interests, academic books and popular fiction, published and unpublished letters and memoranda, school and university essays, among many other kinds of text. The spoken part (10%) includes a large amount of unscripted informal conversation.

While we do not use the raw data in the BNC, we use compiled statistics generated from the BNC. This provides a baseline data model that can be fine tuned using additional corpora.

5.3.4 Corpus of Teenage Language

The Bergen Corpus of London Teenage Language (COLT) is the first large (half a million words) English Corpus focusing on the speech of teenagers [Stenström, 1995]. It was collected in 1993 and consists of the spoken language of 13 to 17-year-old teenagers from different boroughs of London. COLT is now a constituent of the BNC. Statistics

from COLT are expected to exhibit similarity to those from SMS text as the user demographic is similar.

5.3.5 Web SMS Corpus

A raw SMS corpus is maintained online by Jon Stevenson [Shortis, 2000]. This corpus contains over 200 messages. Most of the messages are not well formed English and their authenticity cannot be verified. In line with the extremely personal nature of text messaging, the researcher reports that a proportion of the sensitive and intimate content had to be edited out in the interest of confidentiality. Many messages did not even make it into the corpus because their authors deemed them simply 'too personal' or 'too explicit'. These messages are likely to have been typed on a computer and sent via web based SMS rather than from a phone. This draws questions on the degree to which these messages reflect the language of SMS messaging. Therefore, this corpus is only used for informal testing.

5.3.6 Phrase Sets for Text Entry Research

MacKenzie [MacKenzie, 2003] provides a set of 500 phrases that they have successfully used in the evaluation of text entry methods. While not representative of real world text messaging, it provides a source of sentences that can be used to train Glyph and to compare it with other methods. Punctuation characters are excluded from the text as their addition adds a confounding source of variation that reduces the chances of obtaining statistically significant results.

5.4 Inferences from Corpora

Corpora provide empirical evidence from real language data that can help us understand how text messaging differs from regular communication. Insights obtained from this analysis are then used to design the language model.

5.4.1 Data Preparation

Data preparation involves transforming the corpus into a form that can be readily processed by computer programs. SMS1 is available as an annotated XML file. However, SMS2 is supplied only as a simple web page. Since we do not use all valid characters for our data analysis, both the SMS corpora need to be transformed to use only characters that we are interested in.

Each corpus was subjected to the following transformations.

1. Upper-case characters were converted to lower case. This is common practice in linguistics making it easier to obtain significant results.
2. Accented characters (*café, façade*) were mapped onto their "plain" equivalents (*cafe, facade*).
3. Punctuation marks [, ' " ? . ! @ etc] and space are treated as word delimiters.
4. Contractions are treated as separate words.
5. Numbers are retained.

Each corpus still contains a large number of artifacts like misspelled words and conjoined words without delimiters. These have not been cleaned up and are processed as is, as this has a relatively insignificant effect on character n-grams. After cleanup, SMS1 contains 56 unique characters while SMS2 contains 48.

In addition to a quantitative analysis, we aim to derive qualitative information and inferences from the study of the data sources.

5.4.2 Word Frequency Analysis

There is a marked difference in the type of words that are most frequent in each corpus. SMS1 and SMS2 produce similar lists which contrast with the BNC. The most frequent words in the SMS corpora are those that are commonly used in conversational English, words like ‘I’, ‘Me’ and ‘You’. This can be attributed to the fact that text messages are inherently personal in nature. The degree of similarity in the statistics exhibited by the two SMS corpora reinforces the credence of each as text representative of the domain.

Table 5.3. Ten most frequent words in each corpora

SMS1	SMS2	PSTER	COLT	BNC
i	u	the	you	the
u	i	a	i	of
to	me	is	unclear	and
me	a	to	the	a
at	im	of	nv	in
my	the	you	and	to
go	my	are	it	it
you	at	in	a	is
so	up	for	to	was
can	in	i	yeah	to

Comparing COLT and the BNC, it can be seen that there are differences in the most frequent words. COLT contains teen language and is built from conversational text while the BNC is built from a huge amount of varied English text. This underscores the importance of selecting corpora suited to the task at hand.

5.4.2 Character Frequency Analysis

The SPACE character is by far the most frequent appearing in each corpus at least twice as much as the next most frequent character ‘e’. The SPACE character accounts for almost 20% of characters in SMS1 (21%), SMS2 (21%) and BNC (19%). Character frequencies are strikingly similar across corpora. This shows that despite wide differences in message style, character frequencies in SMS messages are a close match to those in standard English.

Table 5.4. Ten most frequent characters in each corpora

SMS1	SMS2	PSTER	BNC
SPACE	SPACE	SPACE	SPACE
e	e	e	e
o	t	t	t
a	o	o	a
t	i	a	h
n	a	i	o
i	n	s	s
h	s	r	n
r	r	n	r
s	l	h	i

Both SMS corpora contain around 1% of numbers and special characters. This can be attributed to the use of emoticons and shortened words. The most frequent non-alphabetical characters are the numbers 2, 4 and 8 owing to their use in shortening words like tomorrow (2morrow), tonight (2nite) and forget (4get).

Figure 5.1 plots the frequency of each character for SMS1, SMS2 and the BNC. From the figure, it can be seen that the overall character frequencies are similar across corpora. SMS1 and SMS2 are built using expanded character sets and hence exhibit a lower probability value per character. However, the shape of the distribution matches that of the BNC.

Figure 2.1. Character frequencies in corpora

5.4.3 Validity of words

We now check the SMS corpora for the number of valid words. Valid words are defined as those that are present in a dictionary. Spelling checker dictionaries typically contain around 100,000 words while printed dictionaries offer a word count that is an order of magnitude more than 10,000. However, since we are designing for resource constrained devices, the word count of the dictionary must be kept to a reasonable amount.

The dictionary we used for this purpose is the 3esl word list from the 12dicts package [12Dicts, 2003]. The 12dicts package is a collection of English word lists oriented towards common words. The 3esl list attempts to produce an English "core vocabulary" list. It consists of 21,877 words that include abbreviations, acronyms, hyphenations, names and phrases but no inflections.

Table 5.5. Most frequent words not present in the dictionary

SMS1	SMS2
lor	2
ok	im
ur	4
2	ur
wat	darlin
haha	luv
4	wot
dun	txt
la	pete
liao	ok

Table 5.5 shows the most frequently used words in text messages that are not found in the dictionary. Both SMS corpora produce different results owing to the fact that the user demographic that created the messages are quite different. SMS1 was created by Singaporean teens and as such, most of the frequent non-dictionary words such as ‘lor’, ‘la’ and ‘liao’ are words found in Singapore Colloquial English (Singlish). SMS2 is attributed to European teens. Among the most frequent non-dictionary words are the numbers 2 and 4. This is because 2 and 4 are used to represent the words ‘to’ and ‘for’; two words that appear frequently in the corpus and indeed in English.

A signification insight is that 62% of the words in SMS1 and 55% of the words in SMS2 are not found in the dictionary. A portion of these figures are attributed to the lack of inflected word forms in the dictionary used for testing (T9 includes inflected forms in its dictionary). This suggest that dictionary based systems are not very effective for real world text entry. While KSPC for dictionary based systems have been shown to be better than other systems [MacKenzie, 2002], such tests have been conducted using well formed English sentences. More than half of the words in the user’s vocabulary are not present in the dictionary and it requires considerable effort by the user to add these words to his personal dictionary (refer mode switch in T9).

5.4.4 Key level Character Probability Distribution

A mobile phone maps a set of characters to each key. For example, the key D2 (button 2) is mapped to the character set [‘a’,’b’,’c’,’2’]. The character-key mapping is shown in Table 5.6. In the MultiTap system, the user has to press D2 between 1 to 4 times to enter a specific character from the character set.

Table 5.6. Characters mapped to each key

D0				
D1				
D2	a	b	c	
D3	d	e	f	
D4	g	h	i	
D5	j	k	l	
D6	m	n	o	
D7	p	q	r	s
D8	t	u	v	
D9	w	x	y	z

We first determine if one of the characters in the ambiguous character set has a sufficiently higher probability of occurring as compared to its siblings mapped to the same key. We calculate this as follows for alphabets (lowercase, numbers and special characters ignored). We start with the frequency of occurrence of each alphabet in the corpus. For each key, we analyze the frequencies of all the alphabets mapped to that key. This is referred to as the key’s character set.

Figure 5.2. Degree of Disambiguity

We define *Degree of Disambiguity* (DoD) as the difference in frequency for the two most frequent characters in each character set. Higher values of DoD are better as they mean

that the particular key has a higher probability of being used for entering one particular character in that character set more than the others. Lower values increase the ambiguity level as the key is likely to be used for entering multiple characters.

Figure 5.3. Character frequencies for key

Figure 5.2 shows the DoD of each key for SMS1 and the BNC. Figure 5.3 provides more complete data on character frequencies in SMS1.

The DoD for keys D6 and D7 are among the lowest. This observation can be explained by looking at the frequency of characters mapped to these keys. The character set for D7 is [pqrs]. Looking at Figure 5.3, we can see that the characters ‘r’ and ‘s’ are almost equally likely to appear in messages. Thus, key D7 is highly ambiguous.

For most keys, the DoD of SMS1 is slightly lesser than that for the BNC suggesting that the evolution of the language used in SMS messages is driven by the need to reduce ambiguity, which translates into a better text entry experience.

5.4.5 The Effect of Personalization

SMS1 consists of a set of messages contributed by several specific identifiable users and another set of messages collected through anonymous submissions. The two largest contributors to the corpus are user P1 with 745 messages and user P2 with 779 messages. We study each users messaging style to identify useful insights.

Table 5.7. Messaging characteristics for each user

User ID	No of messages	Avg msg length	Avg words per msg
P1(110)	745	45	10
P2(8)	779	62	14

Table 5.8. Most frequent words used by P1 and P2

P1	P2
i	u
u	i
lor	to
ok	lor
2	e
me	so
my	s
then	den
wat	my
now	hmmm

The messaging style is slightly different for P1 and P2. On an average, P2 creates longer messages with more words in each message. In addition, looking at Table 5.9, we can see that both uses focus on a different vocabulary. Of particular interest is the fact that the word ‘hmmm’ appears in P2’s list of most frequent words. This is verified by checking the corpus messages as shown in Figure 5.6.

Figure 5.4. A set of messages created by user P2

5.4.6. KSPC

The Key Strokes Per Character (KSPC) metric was discussed earlier as a method of evaluating text entry performance. KSPC values are calculated for the MultiTap method with the timeout strategy. Values for the MultiTap with NEXT key method have been calculated by MacKenzie to be around 2.032.

KSPC was calculated as follows:

$$KSPC_{MT} = \frac{\sum_{a-z} (K_c \times F_c)}{\sum_{a-z} F_c} \tag{3}$$

where K_c is the number of keypresses required to enter a specific character and F_c is the frequency with which that character occurs in the corpus. Very little variance (1%) has been reported between KSPC computed using character frequency tables and KSPC computed directly from the corpus ('corpus effect').

Table 3.9. KSPC for MultiTap for each corpus

Corpus	KSPC
SMS1	1.66
SMS2	1.65
BNC	1.71

It can be seen the KSPC metrics calculated using the SMS corpora are slightly smaller than that calculated using the BNC. This is due to the differing character frequencies in the corpora. SMS language seems to have evolved so that a body of text as a whole requires lesser keystrokes.

5.5 n-gram Analysis

5.5.1 Word n-grams and their effectiveness

The utility of word n-gram models in predictive text entry has been discussed in Chapter 2. Initial results show that effectiveness depends on the user entering words present in the dictionary. Aside from this factor, storing word n-grams creates a huge challenge as storage requirements for n-grams of order greater than 2 are exorbitant. For example, for a dictionary of 20,000 words, a bigram table would contain 20,000² entries. A trigram

model would require $20,000^3$. Such requirements would quickly overwhelm the limited resources of a mobile device.

5.5.2. Character n-grams and their effectiveness

While there is useful information contained in word n-grams, the sheer number of words in an average user’s vocabulary render higher order n-grams infeasible. Another source of information in a corpus is the character n-gram. Assuming a core set of 27 characters (26 alphabets and a space), even an n-gram of order 4 contains only 27^4 entries. Character n-grams also do not require the user to enter valid words, only valid characters.

Figure 5.5. DoD for unigram and digram models

Figure 5.4 contrasts DoD for a unigram (using simple character frequencies) and a digram model. DoD for the digram model is calculated as follows:


```

FOR Key K with character set C1..Cn
FOR each alphabet prefix Ax where x=a-z
  Find frequency of Ax C1..Ax Cn
    (frequency = SCOREdigram/SCOREalldigrams)
  Normalize
  Sx = Difference in the top two values
DoA(K) = Sx/n

```

Example:

```

For Key D6 with character set M,N,O
For each alphabet prefix Ax where x=A-Z
  Sa = AM, AN, AO
  Sb = BM, BN, BO
  ...
  Sz = ZM, ZN, ZO
DoA(D6) = Sx/26

```

As can be seen, using the additional information contained in the digram results in a major increase in the DoD across the board. This is made clearer by studying a special case. D6 is one of the keys which benefit from the additional information. D6 is mapped to the characters 'm', 'n' and 'o'. In the unigram model, the character frequencies for these characters are m,0.02769903, n,0.05637057 and o,0.06784765 respectively. There is a high probability that the user intends to input either an 'n' or an 'o' using D6 in his text entry process. However, the addition of a prefix in the bigram model acts to limit the number of choices of characters that can follow. As can be seen from Figure 5.5, once the user has entered 'a' and pressed D6, it is highly likely that the next character will be 'n' rather than 'm' or 'o' thus reducing ambiguity.

Thus, character n-grams contain useful information which if used effectively, can increase text entry rate.

Figure 5.6. Best character match for key D6

5.6 Limitations of corpora analysis

While SMS1 provides a large quantity of messages, the content of real world messages is sometimes too rich to process accurately. Several compromises have to be made to enable a system to generate statistics. For example, only lower case processing is performed and several special character sequences are ignored.

SMS2 contains raw messages and the credibility of the data collection process cannot be ascertained. However, statistics for SMS2 match closely with that of SMS1 suggesting that SMS2 is indeed a representative source of SMS text.

Additional inferences can be made if the corpus includes conversations rather than distinct messages. The addition of even more messages also increase the accuracy of models trained using this data.

5.7 Design Goals for Glyph

The primary design goal of Glyph is to provide a better text entry experience for the user. This translates to providing a fast and easy to learn system that provides accurate predictions while minimizing cognitive loads. The major factors guiding the design are examined below.

5.7.1 Learning Curve

The average consumer is highly averse to change. He is comfortable with systems that he is used to and it will take a lot of convincing before he switches to a new way of doing things. A good case in point is the failure of the Dvorak keyboard to gain consumer acceptance despite it being more efficient than Qwerty.

A new text entry system should not require a significant learning curve. Many new systems, like Thumbscript and Q12 (described in Chapter 3) require the user to learn radically new ways of inputting text.

5.7.2 Accuracy

Accuracy refers to the ability of the system to disambiguate the user's key presses successfully. Ambiguity is difficult to overcome as more than 30 characters have to be mapped to 12 keys. As will be discussed later, improving accuracy comes at a cost.

5.7.3 Flexibility

SMS is primarily a means of inter-personal communication. A text entry method should not artificially constrain the user forcing him to change his messaging style. He should be free to be as expressive in his communication as the medium allows. A dictionary based system tends to force the user into entering well formed English sentences, one of the reasons why many users have abandoned such systems [Fait et al, 2002].

5.7.4 Cognitive Load

The cognitive loads involved in mobile text entry have not been studied in detail. We believe it is of crucial importance to develop models to understand the cognitive loads imposed by a text entry method. For example, while T9 can theoretically achieve a high text entry rate, it has at least two major drawbacks that create a high cognitive load on the user. Firstly, words being composed can change wildly from key press to key press and T9 tends to 'blow up' in the face of errors. Secondly, adding new words to the dictionary require the user to put the text composition task on hold, break the communication and switch to another text entry mode.

5.7.5. Adaptiveness and Personalization

Systems that learn user behaviour can provide a better text entry experience if implemented correctly. As can be observed from the corpus studies, SMS users use a highly personal vocabulary that differs from a standard English vocabulary, even varying from user to user. While initial systems provided little in the way of learning, newer systems support extraction of user data from disparate sources and explicit addition of user words into the system dictionary. However, there are few systems that learn transparently without user intervention.

5.7.6 Resource Constraints

The capabilities of mobile devices in terms of processing power and storage have been steadily increasing. However, mobile devices are still extremely resource constrained. A text entry method has to make a tradeoff between accuracy and resource usage. Even in terms of resource usage, there is a tradeoff between application response time and memory utilization. For example, the internal structure used to store the data can be optimized either for fast lookups or for a minimal memory footprint.

With these goals defined, we now look at the design of Glyph.

5.8 Glyph Architecture

5.8.1 Message Composition

Predictive systems differ from their nonpredictive counterparts primarily in the way they deal with message composition. Figure 5.7 (modified from the Reactive Keyboard) shows the message composition process in Glyph.

Users decompose their intended message into a series of selection elements. Each element is then selected using designated keystrokes (*composing* mode). Selection elements can include character lists, word lists or possible word completions (*lookahead*). The system then synthesizes the series of selection elements back into the user's intended message (*committed* text). In non-predictive systems, a fixed set of elements that are equally and deterministically accessible are presented to the user. This set of selection elements and its ordering never change. A predictive system on the other hand uses prediction rules to make likely elements easier and faster to select.

Figure 5.7. Message Composition using Predictive Text Systems

The system monitors the user selections and stores a selected number of selection elements in its short-term memory. This forms the current context. Long-term memory is then consulted for information on this context to find possible predictions. Depending on the display policy adopted by the system for displaying possible candidates, the prediction engine generates predictions that are then displayed by the visual system.

5.8.2 Prediction Engine

The prediction engine is the heart of the predictive system and performs the key function of generating predictions. Glyph takes a hybrid approach in the design of its predictive core. Statistical character prediction is used as the default input method. This is augmented by a dictionary based word completion system. Improved versions of Glyph can be designed by augmenting the character prediction engine using word n-gram statistics.

Character Prediction Model

The character prediction in Glyph is based on an n-gram model supporting variable length prefix matching. The long term memory of Glyph is initially modeled in the form of a modified *trie* [Fredkin, 1960]. The word *trie* comes from the word ‘retrieval’ as tries are a popular data structure in information retrieval systems.

Model Construction

The model is constructed by priming with a large number of representative text samples - the corpus - to establish the relative frequency of n-grams in the language of interest. The absence of a good SMS corpus has been highlighted by many researchers in the field as a major factor holding back text entry research. This problem is exacerbated in the context of the project as we use an n-gram model rather than a k-gram model. This implies that the text samples used for training must be in the form of individual sentences; simple word lists are insufficient.

An example of the conceptual structure of long-term memory after priming it with the text “thanx aton” (Thanks a ton.) is shown in Figure 5.2. The memory represents an n-gram model of order 3 that includes all lower order models and naturally conforms to a tree structure. ‘#’ is the null context and the root of the tree. The very first level of the tree corresponds to a model of order 1 or a unigram model. When primed over a sufficiently large sample of text, this basically represents individual character frequencies of the language under study. Nodes on lower levels provide increasing amounts of information about the context that it belongs to. Looking at the leftmost ‘T’ we can infer that the character ‘T’ is typically followed by an ‘H’ or an ‘O’ (inference based on primed text). Deeper in the tree, we can see that the context ‘TH’ is succeeded by an ‘A’.

The goal of the structure is to facilitate fast matches with the context contained in short-term memory. In addition to a character label, each node has frequency information that stores how many times that particular context has been seen. The branching factor of the tree in the example is very small owing to the short length of the text used to create it. In actual scenarios, the tree usually branches out at maximal capacity for n-gram models of English of relatively smaller orders (<3). The implementation of long-term memory is discussed in Chapter 6 along with a character by character walkthrough through the model construction process.

Generating Predictions

Generating a prediction involves walking the model tree based on the contextual information present in the short-term memory. Consider the prediction suggested by the system after the user has typed the following text “thanx at”. The n-gram model we have

generated earlier is of order 3. Hence, we select the last 2 characters of the user input, ‘at’ and use that as the current context. Looking at the tree, we can see that the only character that has been encountered after this context is ‘o’.

Figure 5.8. Long Term Memory Structure

In the event that a context has not been encountered in representative text, the system reduces the context by 1 character and tries to look for a match in the (n-1) order n-gram model. For example, if the context is ‘tx’, we can see by looking at the model that no such context has been seen by the system. In this case, the context is shortened to “x” and a match is attempted again. This is successfully as a context “x” exists in the model with a prediction of ‘ ‘.

The Reactive Keyboard, owing to its specific goals, goes one step further in its prediction generation. It tries to generate as many single character predictions as possible using the model. Each such prediction is then added to the current context and the new context is used to generate another prediction. This process continues until a system defined rule (‘Stop when a space character is encountered’) is reached. Thus, the Reactive Keyboard tries to arrive at a large number of predictions, some even in the form of full sentences. However, the reason for this is that the Reactive Keyboard is designed primarily as a tool for the physically disabled where minimizing motor activity is a very high priority. Expansive predictions like this require very high order n-gram models to be effective. This translates into heavy memory requirements and processing power, a luxury not afforded to the mobile user. Therefore, Glyph restricts itself to smaller order n-gram

models and single character prediction. Alternative approaches are taken to provide word completion features (refer Section 5.4.2).

5.9 Techniques Implemented to Improve Text Entry

This section highlights key design decisions taken in the design of Glyph to improve text entry in terms of text entry rate and subjective user satisfaction. The resource limited nature of mobile devices is a major hindrance in the application of advanced algorithms and methods. Memory footprint has to be as small as possible while computational complexity of a predictor has to be minimized both for increased response times and to conserve power.

5.9.1. Bridging Word Boundaries

The LetterWise system is based on a selective subset of prefixes of order 3. Individual words in the text are typically separated by the SPACE character. When performing the a priori statistical analysis of text corpora, LetterWise does not take into account the context before and after the SPACE character. In Figure 5.9, the context of order 3 surrounding the SPACE between the words ‘See’ and ‘You’ is ‘E_Y’. The equivalent context between ‘We’ and ‘Want’ is ‘E_W’. It has been noted that in character prediction systems, the first letter of a new word typically has maximum ambiguity due to the absence of any prefix information for the null context. This is also known as the *zero frequency* problem. Approaches to overcome this problem include providing the character that has been seen the most number of times in representative text or by just picking a character with the highest rank in the collection.

Figure 5.9. Crossing Word Boundaries

Glyph tries to acquire information across word boundaries by including the contexts centered on the SPACE character. Thus, the zero frequency problem occurs only when typing the very first character of the entire text. In this case, we display the most frequently seen character as the likely candidate.

5.9.2. Expanded Character Set

Typical text messages contain emoticons made up of numbers and punctuation characters. In its current version, the internal model of Glyph takes into account basic punctuation symbols. A tradeoff of this decision is the increased memory footprint of the data model. Table 5.10 shows the tradeoff between the richness of the character set considered and the memory footprint of the model for an *n*-gram model of order 3.

Table 5.10. Effect of Model Character Count on Memory Footprint

Characters in the model	Memory Footprint (units)
26 (alphabets)	18,278
37 (+numeric+space)	52,059
45 (+basic punctuation)	99,498
60 (+rich symbol set)	219,660

One solution to the problem of high memory requirement is to design the model to *forget* rarely used entries. Each time a unique context is seen and added to the system, the least frequently used node is deleted from the system. This adds significantly to the complexity of the system particularly in persisting the internal data structures.

5.9.3. Effect of Context Length

The data model of the system is modeled as a trie. It has been discussed before that increasing the number of characters that make up the context increases the accuracy of the prediction. However, memory requirements for a trie grow exponentially with increasing depth. Table 5.11 shows the tradeoff between context length and the memory footprint of the model for a 26 character set.

Table 5.11. Effect on Context Length on Memory Footprint

Context Length (number of characters)	Memory Footprint (units)
1	26
2	702
3	18,278
4	475,254
5	12,356,630
6	321,272,406

Several optimization techniques have been proposed to compact trie data structures. Optimization is not a major focus of the project and hence will not be discussed in detail in this report.

5.9.4 Word Completion

The KSPC of a statistical character prediction system has a maximal value of 1 which happens with 100% accurate predictions because each keystroke is disambiguated to the intended character (this is an ideal situation and is typically not achieved). To bring down KSPC even further, Glyph adds an internal dictionary to its model. Whenever a context matches a prefix in the dictionary and a *confidence parameter* is satisfied, the system suggests a word completion. The confidence parameter is a measure of how confident the system is that the suggested completion will be accepted by the user. This parameter can be calculated based on a number of factors including possible benefit to the user (keystrokes saved) and frequency of correct predictions in the past. Foster [Foster et al, 2002] describes a confidence calculation method that can be adapted for use in Glyph.

Word completion provides maximum benefit when the correct word is of above average length and is predicted early. This results in maximum keystroke savings. The nature of text messaging has been discussed earlier and it was noted that the evolving language was composed mainly of shortened words. This might reduce the usefulness of adding word completion to the system. However, corpus studies reveal that the average word size in SMS messages is not substantially lesser than that of normal English text.

Conceptually, two separate dictionaries are used by the system. One is a predefined dictionary of size n that contains the most commonly used words found in representative text. The other is a personal dictionary of k words that were learnt through the automatic modeling process. A key decision is the size of n and k . However, this can be fixed only after the character prediction model has been optimized and the remaining memory resources are audited. The vocabulary of an average human has been estimated at around 10,000 words. Typical phone conversations use words from a set of around 5,000 [Dunlop et al, 2000a].

5.9.5. Learning and Adaptation

A *user model* is a knowledge source which contains explicit assumptions on all aspects of the user that may be relevant for the behavior of the system. A *user modeling component* is that part of an interactive system whose function is to

- incrementally build up a user model,
- to store, update and delete entries in it,
- to maintain the consistency of the model,
- to supply other components of the system with assumptions about the user.

Techniques for user modeling in interactive computing can be classified according to whether they place the burden of modeling on the designer, the user, or the machine. Four classes can be identified:

- *canonical* modeling, where the designer of the system models the user
- *explicit* modeling, where users model themselves
- *automatic* modeling, where the system models the user
- *combination* modeling, which fuses the automatic and explicit methods

The boundaries between the different systems are blurred and well designed systems typically use a combined approach.

The user modeling strategy adopted is a critical aspect of Glyph and is perhaps its most unique feature. Modeling is canonical when the designer provides a language model for the user. While simple and easy to implement, it has the disadvantage of being inflexible to user needs. Canonical models are developed based on designer intuition, analysis of representative text, analysis of user tasks and participatory studies involving users. The final model is a frozen representation of the requirements of a typical user and not of the individual. It is thus difficult to make a case for canonical modeling for mobile text entry methods. It has already been identified that the 'language' of text entry varies from person to person and that a general purpose model is ineffective.

Explicit modeling involves the designer providing options to the user to personalize the system. Any change the user makes to the system in order to influence its functioning

becomes a part of the model. The T9 system uses a combination of canonical and explicit modeling. In the event that the word intended by the user is not available in the internal model, the system prompts the user to enter and add the new word to the model. The problem with this method is that in practice, most users are likely to use the default settings due to the complexity of explicit modeling (explicit modeling in T9 is discussed in Section 3.2). The model thus defaults to a canonical one.

Modeling is automatic when the system models the user. Automatic systems construct their model from scratch by continually monitor user behavior and updating the internal model. General purpose automatic modeling is likely to be inaccurate as it is difficult to interpret all user actions. In addition, user errors also find their way into the model as part of the learning process. However, an advantage of automatic modeling for a mobile text entry system is the transparent way in which the system adapts to the user. Unlike the T9 system, there are no cognitive disruptions when the word intended by the user is missing from the model. The user continues typing as usual and once the word has been entered, the system transparently adds it to the model thus ensuring that the word is available for prediction the next time a valid prefix is entered.

The learning system in Glyph comes into play each time the user composes a message. Real-time updates to the model after each character is entered are likely to result in poor system response times. Hence, updates are deferred till after the entire message is composed and committed. The priming algorithms used to build the initial model are then invoked to process the message and update the model. Character frequencies within their contexts are incremented and any unique words identified in the message are added to the dictionary. Thus, although Glyph starts out with a canonical model, it uses automatic modeling to adapt itself to the user's style of text composition.

Chapter 6

Implementation

The challenges faced by developers in implementing and testing new mobile text entry method have been highlighted in Chapter 4. The prototype implementation of Glyph is developed using the Prototyping Framework. Java is chosen as the development language for ease of integration with the framework and for code reuse advantages when porting the over system to mobile devices.

The final mobile device ready version was implemented in C# using the Microsoft Smartphone 2003 SDK. The application can be deployed on any Smartphone device.

6.1 Prototype Development

The MultiTap and T9 systems were implemented first to test the prototyping framework and to get a feel for the issues that can arise in the development of text entry methods. A brief description of the MultiTap implementation is provided in Section 4.6 while the T9 implementation is discussed in Section 4.5.

6.2 System Architecture

We begin with an overview of the system architecture followed by a discussion on the data structures used to store the model. Figure 6.1 shows the important components of the system.

An application running on a powerful desktop computer (Figure 6.2 and 6.3) is necessary in order to process corpus data and generate the data structures. The compacted data structure is then uploaded to the mobile device. The model is a fairly complex structure that needs to be carefully studied throughout the development process. Several model analysis tools were developed for performing tasks ranging from visualizing the model contents, validating the model, calculating memory footprint and generating statistical information.

Figure 6.1. System Architecture

6.2.1 Priming Subsystem

The Priming subsystem is responsible for converting raw text samples into a form usable by the predictive mechanism. Raw text content is cleaned up depending on the current testing rules (e.g.: strip all punctuation characters) and the model is created. The priming system processes text one sentence at a time so that context is maintained. It currently does not take into account the often session-based nature of text messaging.

Figure 6.2. Glyph Data Analysis Tools

The trie management tools support:

1. Creation of order n tries
2. Training tries using preprocessed sentence and word lists
3. Transforming raw corpus files into processed sentence and word lists
4. Saving tries to compact binary files
5. Saving tries to XML
6. Loading tries from binary files
7. Visualization of the tree structure of the trie
8. Generating statistical information

6.2.2 The Language Model

The model is stored in the form of a suffix trie. A basic trie of size (h, b) is a tree of height h and branching factor b . All keys can be regarded as integers in the range $[0, b^h]$. Each key K can be represented as an h -digit number in base b : $K_1K_2K_3\dots K_h$.

Figure 6.3. Trie Management Tools

The trie is implemented in the form of a tree structure made up of `TrieNodes`. Each `TrieNode` contains a `Hashtable` of b `TrieNodes` with individual characters being used as the keys. The `Trie` object contains a single reference to a root `TrieNode`. In order to conserve memory, `Hashtables` within `TrieNodes` are created using *lazy instantiation*, a method in which the data structure is initialized only upon first access. This ensures that the system does not create a wasteful maximal tree structure when initialized.

Each leaf node also contains a list of words that start with the word stem that matches the path to the leaf node. This is an area for open to optimization. *PATRICIA* tries [Szpankowski, 1990] can be modified to compact the data structure and avoid duplication of data.

Figure 6.4 The node data structure

6.2.3 Building the Model

Once a trie has been created, it can be trained multiple times using preprocessed data. Each training run consists of two passes. The first pass processes individual messages to update the n-gram model. The second pass processes a word list to augment the n-gram model with word completion capabilities. Although the word list is typically generated from information in the message list, it can also be a custom dictionary.

Figure 6.5 shows the step by step procedure for building the character model (n-gram model of order 3). We start with an empty model. The training system is called with the message “can u call” (‘Can you call?’). From Step 1, we can see that the initial context ‘can’ is processed as three contexts ‘can’, ‘an’, ‘n’. The system also supports training with single contexts. Nodes are created (Create Action) as necessary and if a node is already present, its frequency component is updated (Update Action) (updated nodes are shown in gray). As more and more text is processed, the trie becomes denser and branches out to almost full capacity for lower order models.

Once the n-gram has been updated, the word list is processed to augment the model with word information. Only a fixed number of words are allowed (4 in the current implementation) to reduce memory requirements. Words are added to a node that already contains 4 words only if the frequency of the new word exceeds the frequency of the least frequent word in the list. The most frequent words in the data source are given priority.

Step	Context	Action	Model
1	can	Create CAN
	an	Create AN	
	n	Create N	
2	an_	Create AN_
	n_	Create N_	
	_	Create _	
3	n_u	Create N_U
	_u	Create _U	
	u	Create U	
4	_u_	Create _U_
	u_	Create U_	
	_	Update _	
5	u_c	Create U_C
	_c	Create _C	
	c	Update C	
6	_ca	Create _CA
	ca	Update CA	
	a	Update A	
7	cal	Create CAL
	al	Create AL	
	l	Create L	
8	all	Create ALL
	ll	Create LL	
	l	Update L	

Figure 6.5. Model Construction

The priming process is both CPU and time intensive; in fact it can take many minutes to build a model from a large quantity of text. Thus, it is not practical to recreate the model each time a message needs to be composed. To overcome this problem, the priming

system *serializes* the model to a persistent format that can then be loaded by the prediction system on startup. To achieve the lowest file sizes, the data structure is serialized to a binary file using a custom data format. A word augmented trie of order 3 created from SMS1 contains 10,042 nodes. The serialized binary file is 127KB in size while the equivalent data structure in XML format takes up 1.4MB. Compressing the binary file reduces the size to a mere 46KB.

6.3 Generating Predictions

The core function of the predictor is to generate accurate predictions. The prediction engine is queried each time the user presses a key. Keys are divided into two classes, input keys and control keys. Input keys include keys that are used to enter character data that is added to the message. Control keys provide additional functionality like deletions, clearing text, selecting predictions etc.

Figure 6.6 The DataManager class

Once the user presses a key, the character set mapped to the key is sent to the prediction system (represented by the `DataManager` class) along with the current text context. Using the current contextual information, the predictor builds a context of size 2 (assume model of order 3) which is the last 2 characters of committed text. The aim of the character prediction system is to disambiguate between the characters mapped to the button that was pressed. If the user has typed “can u c” so far and now presses the D2 key on the phone keypad, the system has three possible candidates (assume numbers are

excluded from the model), namely ‘a’, ‘b’ and ‘c’. The predictor creates three new contexts (‘_ca’, ‘_cb’, ‘_cc’) by appending each possible candidate to the current context ‘_c’. The model is now queried for the nodes corresponding to these three contexts. This involves walking through the trie using the context as the path information. Depending on the scores calculated for each of those characters, the system returns the most likely character to follow in the context.

Word predictions are handled in a different fashion. The word predictor works on the same data as the character predictor. It also queries the same data model. However, rather than using the entire context, it extracts the current word fragment from the context. The space character is used as a delimiter to detect the beginning of a new word. In the current version, word predictions are only supported for word stems of size (n-1) where n is the order of the data model. For example, if presented with the context in Figure 6.7, the predictor extracts the last word stem (‘to’). It combines this string with each of the characters ‘m’, ‘n’ and ‘o’ in the key D6 that the user pressed to generate three lookup keys ‘tom’, ‘ton’ and ‘too’. The language model is queried for all stored words that match these stems. The results of the query for each key are combined into a single word list which is then sorted in the order of their scores. The top 4 candidates are then returned to the display system for selection by the user.

Figure 6.7. Predictions in Glyph

6.3.1 Handling incorrect predictions

It is difficult to provide accurate predictions all of the time. If the user intended a particular character and the predictor made an incorrect guess, the system allows the user to cycle to his intended character without imposing additional artificial steps. In the

current version of Glyph, character orders within each character set are not changed so as to not confuse the user. If the initial predicted character is wrong, the user can cycle to the correct character in the sequence by pressing the same key multiple times. An alternative approach and possibly a more efficient one is to reorder the character set based on character scores. However, this can lead to character sets changing their order on each key press imposing high cognitive loads. This issue and a possible solution are discussed further in Section 7.5.

6.4 Displaying Predictions

All display elements including text and graphics are custom rendered by the display subsystem of Glyph. The display is composed of the following aspects:

6.4.1 Composing Mode

The user is said to be in composing mode when he has pressed a key but not confirmed which character on the key he has intended. The current character is drawn underlined and in a lighter color. Once the user has committed his selection, either by way of timeout or an explicit timeout, the new character is added to the main text which is displayed in black. The first letter of the sentence is automatically capitalized as is common in current text entry systems.

6.4.2 Character Status

The best match character in a key's character set can change depending on the current context. To reduce confusion, a clear indicator is provided in the bottom left corner as to which character is currently selected and the order in which characters can be cycled.

The first letter in each word is the most ambiguous. Additional cues can be provided to help the user make a selection. For example, EziText display a small triangle with all the elements of the key's character set at the point of insertion.

6.4.3 Word Prediction List

The word prediction list is a source of visual noise in the composition process. As such, it is designed to reduce its intrusive nature. Only a maximum of four predictions are displayed at any one time and predictions are only attempted if they pass certain rules. This acts to display word predictions only when the word is sufficiently large and likely and therefore, is expected to save key strokes.

Human computer interaction theory suggests that short term memory can hold around 7 elements comfortably [Card et al, 1983]. Considering the fact that screen space in a mobile device is at a premium, we arrive at a consensus figure of 4 elements in the selection list. The selection items are sorted according to their frequency and positioned correctly below the word stem. They are also rendered in a gradated color scheme in which the more frequent element is drawn in a darker color. An improvement would be to display the first prediction inline with the text.

6.5 Learning

Glyph is a transparent learning system. Because of the modularity of its design, the priming subsystem can be reused to provide runtime learning capabilities. Messages typed by the user are processed after they are completed and committed. Adding training data is a resource intensive task and deferred updates allow Glyph to maintain good response times. In addition, training only after completion ensures that no erroneous words are inserted into the model. After each message is completed by the user, the system processes the message and updates the model with the character and word data in the message.

6.6 System Setup

The prototyping framework is configured to display a generic mobile phone. The numeric section of the keyboard is remapped to the mobile phone keypad layout. The keyboard keys are also physically relabeled with printed labels as shown in Figure 6.8 to enable more efficient testing and development.

The mobile version of Glyph is designed for Microsoft Windows powered Smartphones and has been tested on the Smartphone emulator. A Windows Mobile-based Smartphone integrates PDA-type functionality into a voice-centric handset comparable in size to today's mobile phones. Windows Mobile-based Smartphones are designed for one-handed operation with keypad access to both voice and data features. They are optimized for voice and text communication and come with at least 16 MB of RAM and a large color screen. The Motorola MPx200¹, a typical Smartphone, provides a 176 x 220 pixel color TFT display and 32MB of Flash ROM supporting up to 1GB of pluggable memory.

¹ <http://commerce.motorola.com/consumer/QWhtml/mpx200.html>

Figure 6.8. Remapped keyboard

The data model is generated by the desktop application and then deployed to the phone using ActiveSync. Early versions of Glyph took almost 20 seconds to load the data into the phone memory on startup. This was traced to inefficient methods for loading the data. The improved version loads a trie of order 3 containing over 10,000 nodes and over 4,000 words in less than 1 second.

Figure 6.9. Deploying Glyph

Chapter 7

Evaluating Glyph

The assessment and comparison of a new text entry method with current methods is a necessary part of the design process. The best way to do this is through an empirical evaluation. Unfortunately, such evaluations are time-consuming and complicated. Careful planning and execution is required when undertaking such an evaluation, as an abundance of confounding factors exists that could negatively affect its repeatability and validity. We focus on a discussion on the KSPC metric [MacKenzie, 2002b] that is typically used to compare current text entry systems. Key Strokes Per Character (KSPC) is the number of keystrokes required, on average, to generate a character of text for a given text entry technique in a given language. KSPC is usually calculated using a linguistic model.

This evaluation compares Glyph with the two most popular and established text entry systems, T9 and Multitap. We compare the three systems based on several factors including KSPC, text entry rate and cognitive load. We also compare them using both well-formed sentences and real world messages. Glyph is studied in particular detail to evaluate the effectiveness of the system depending on factors like corpus type and n-gram order.

Review of Corpora

The NUS SMS Corpus [How, 2004] is a corpus of SMS messages (SMS1) collected for research at the Department of Computer Science at the National University of Singapore. The corpus consists of about 10,000 SMS messages, largely originating from Singaporeans and mostly from students attending the University. These messages were collected from volunteers who were made aware that their contributions were going to be made publicly available. The bulk of the corpus (60%) is formed by messages collected on a regular basis from several specific phone users aged between 18 and 22 years. The

remaining content of the corpus is composed of web based submissions. The two largest contributors to the corpus are user P1 with 745 messages and user P2 with 779 messages. Message samples from SMS1 are shown below.

hey so whats the plan this sat
can call me at 10:10 to make sure dat i ve woken up
hey i will be late ah meet you at 945+
hey tmr meet at bugis 930

The British National Corpus (BNC) is a very large (over 100 million words) corpus of modern English, both spoken and written [Clear, 1993]. The Corpus is designed to represent as wide a range of modern British English as possible. A raw SMS corpus (SMS2) is maintained online by Jon Stevenson [Shortis, 2000]. This corpus contains over 200 messages. Most of the messages are not well formed English.

MacKenzie [MacKenzie, 2003] provides a set of 500 phrases (PSTER) that they have successfully used in the evaluation of text entry methods. The well-formed sentences chosen for evaluation are the phrase sets in PSTER. As can be seen below, these sentences are not representative of real world text messages.

my watch fell in the water
prevailing wind from the east
never too rich and never too thin
breathing is difficult
i can see the rings on saturn
physics and chemistry are hard

7.1. MultiTap

MultiTap has been discussed extensively in Chapter 3. KSPC values are calculated for the MultiTap method with the timeout strategy in Section 5.4.5. Very little variance (1%) has been reported between KSPC computed using character frequency tables and KSPC computed directly from the corpus ('corpus effect'). For a linguistic model based on SMS1, KSPC was found to be 1.66.

Using the more popular NEXT key method, KSPC for MultiTap has been calculated to be 2.032.

7.2. T9

KSPC for T9 has been calculated at 1.0072 [MacKenzie, 2002b]. However, this calculation assumes that all words are found in the dictionary. In addition, it does not take into account key strokes required to select candidate predictions or to correct errors. Due to the problems plaguing the T9 system (Section 3.2.2), the number of such key strokes is likely to be a sizeable amount.

T9 uses an encrypted dictionary file and there is no public information about the content of its internal dictionary. We are thus unable to get an accurate value of KSPC for T9 using representative corpora like SMS1 and SMS2. However, we can make an estimate using a standard dictionary of English words.

As discussed in Chapter 6, only 37.99% of the words in SMS1 can be found in the dictionary. T9 defaults to MultiTap mode for entering non-dictionary words. A more realistic value of KSPC can be calculated as follows:

$$\begin{aligned}\text{KSPC} &= (\% \text{ of dictionary words}) * \text{KSPC}_{\text{d}} + (\% \text{ of non-dictionary words}) * \text{KSPC}_{\text{mt}} \\ &= 37.9896 * 1.0072 + 62.0103 * 2.0342 \\ &= 1.6440\end{aligned}$$

However, this value does not take into account the fact that the dictionary used for testing does not contain inflected words. It also does not take into account the fact that users can add words to the dictionary. KSPC will be lesser when these factors are taken into account and the value can be expected to move closer to the value for 1.0072.

7.3. Glyph

7.3.1. Calculating KSPC of Glyph

We begin by selecting an n-gram order and a data source to train the system. A set of test messages is prepared for testing. Due to the scarcity of valid messages, the data source used to train the system is used for testing purposes. Each message in the test set is passed to the Glyph runtime system. The testing framework uses the following algorithm to arrive at a KSPC value.

Messages are processed character by character. For clarity, we only consider a limited character set comprising of alphabets and Space. The mobile version of Glyph adds number support while the data model supports an arbitrary number of characters. The key

corresponding to the message character is retrieved. The current context is the portion of the message seen so far. The current character set is the entire set of characters mapped to the specific key we have retrieved. This forms the input to the prediction mechanism of Glyph.

```

FOR each Character C in the Message
  Get the key K that maps to C
  Get the character prediction list CI suggested by Glyph
  Keystrokes required is the index of C in CI
  (A successful prediction means that C is the first letter in set CI)

  Get the word predictions WI for the current Context
  Lookahead in the Message for the current word W
  IF W is found in WI
    IF selecting W saves more keystrokes
      Update keystrokes
      Advance character pointer to the start of the next word in Message
    ENDIF
  ENDIF
  Add C to the current Context
ENDFOR

```

The predictor returns the character set sorted in the order of each character's score. Typically, the highest ranking character at the top of the list will match with the message character. In this case, the number of keystrokes required is only one. In the event of a mismatch, the user cycles to the correct character in the list. Thus, the total number of keystrokes is one plus the zero based index of the character in the list. The current character is added to the context and the process is repeated for each character of each message in the data set.

Word predictions add further complexity to KSPC calculations. Selections are more complex and the prediction triggers have to be decided beforehand. In the current version of Glyph, words are only stored in the leaf nodes of the data model. This means that the length of the key or the word stem used to lookup these entries need to be equal to the n-gram order. Hence, this does not give a completely accurate view of the benefits of word prediction. For each attempted prediction, Glyph displays from 0 to 4 choices. The user uses the joystick or designated keys to select a particular entry from the list and another key to accept the selection. This adds the word to the message along with a leading Space character. Selecting the first predicted word requires only a single key press to commit it.

On the other hand, selecting the last word requires three key presses to navigate to the word and one to select it leading to a total of four key presses. We also assume that the user always tries to minimize the number of key presses. The user skips the word predictions offered if the number of key presses required to enter the word using the character prediction model is less than that incurred by selecting the word through word prediction. This typically occurs when the predicted words are small or if there are a large number of ambiguous words. This problem is partly reduced in Glyph as word predictions are performed only after certain criteria are satisfied, one of which is sufficient word length. In addition, this problem can be further ameliorated by efficient selection of word stem length. Longer stems tend to produce more accurate word predictions at the cost of a higher number of key presses.

Word predictions accepted by the user are added to the context along with a Space character and the keystroke counter is updated. Once all the messages have been processed, the number of keystrokes is divided by the number of characters to arrive at a KSPC value for Glyph specific to the training set, test set and n-gram order.

7.4. KSPC Comparison

Figure 7.1 compares the KSPC values for Glyph (trained on PSTER), Glyph (trained on SMS1), T9 (trained on SMS1), T9 (perfect) and MultiTap.

Figure 7.1. Comparing KSPC

Values are plotted for n-gram order ranging from 1 to 5 without enabling word prediction. As can be seen, while 1-gram models of Glyph perform on par with MultiTap, utilizing the information contained in higher order n-grams brings down the KSPC of Glyph to a low of 1.071. While it will be shown later that higher order n-gram like 5-grams are not feasible to deploy, their performance can be seen to be slightly better than even the best case estimates of T9. The KSPC curve for Glyph flattens out for higher values of n suggesting diminishing returns on higher order n-grams.

Figure 7.2. Testing with different data sources

Figure 7.2 shows the results of testing Glyph using several available data sources. The system is trained using SMS1 with an n-gram of order 4. As can be seen, there is very little variance for the character model across different test sets. PSTER scores slightly higher owing to the well-formed nature of its sentences. P1 and P2 contributed relatively fewer messages to the test set. It can be seen that the character information mined from the training data is broadly applicable across different messaging styles. We now investigate how word prediction affects KSPC values.

Figure 7.3 compares KSPC values for versions of Glyph augmented with word prediction. Most noticeable is the fact that even 1-grams perform far better than their purely character prediction counterpart. This is expected as word completion saves a

significant number of keystrokes depending on the accuracy of the prediction and the length of the predicted word.

Figure 7.3. KSPC with word prediction

Figure 7.4. The effect of word prediction

Comparing version of Glyph with and without word prediction (Figure 7.4) highlights the impact of word prediction on KSPC. The best case KSPC of 0.972 is achieved with a 3-gram with word stems of length 2. As mentioned before, currently, word stem size is

bound to n-gram order. This factor is attributed to the *increase* in KSPC after order 3 owing to the increased word stem size and the fact that the average word length in the data set is lesser than that found in common vocabulary.

We now compare the KSPC achieved with the cost of achieving it, namely in terms of memory required. Figure 7.5 plots KSPC for each n-gram order against memory requirements accrued by the data model (uncompressed and compressed). As can be seen, an n-gram of order 5 requires over 1MB of memory uncompressed and over 200KB when compressed. This is only in terms of static disk cost. Loading this structure into memory will incur significant costs unless heavy optimization is preformed. This can inversely affect prediction times. A good balance can be achieved with an n-gram order of 2 or 3.

Figure 7.5. The tradeoff between KSPC and model size

7.5 Improvements in Text Entry

The text entry rate is enhanced using a combination of improved disambiguation of the current character using techniques outlined earlier, combined with the word prediction system. For example, the prediction of 'meeting' after the user presses 'm' results in a saving of at least six keystrokes (six is the best case as the MultiTap system can be expected to require many more keystrokes with T9 faring slightly better). However, this is based on the assumption that the predicted word is what the user intended. The accuracy is dependent on the language model, effectiveness of the learning system and

the confidence filters. Given an accurate system, the text entry rate is definitely improved.

The nature of mobile text messaging has been discussed in Chapter 6. It describes a personalized messaging style made up of short words and special characters. Current predictive systems are unable to handle this level of personalized texting with the exception of MultiTap. MultiTap being completely unambiguous is well suited for such a task. However, MultiTap requires the most number of keystrokes when compared with other text entry systems and does not offer word completion capabilities. Glyph provides the next best alternative by introducing controllable character level ambiguity to reduce the number of keystrokes.

The segmentation problem in MultiTap (Section 3.1.1) is also present in Glyph. For entering consecutive characters mapped to the same key, the user needs to adopt one of the three segmentation strategies outlined earlier. However, effective word prediction can mitigate this issue to some extent.

Figure 7.6. Entering personal lingo in Glyph

An interesting side effect of the learning process is that by default, the system learns number sequences. As shown in the first two screens in Figure 7.7, if key D8 is pressed after entering the number 9, the system suggests the number 8 as a possible completion. If the user intended to enter a mobile phone number, as is often the case, this saves either the effort required to switch modes to number input or the large number of keystrokes required to enter numbers in text mode. A possibly undesirable side effect is that the system adds numbers (typically not reused) to its dictionary.

Figure 7.7. Side effects of learning

Reduction of Cognitive Load

The MultiTap system has minimal cognitive load. The user is completely in control of what data is entered and there are no ‘surprises’. As mentioned before, the drawback is the substantial increase in keystrokes required to enter text. The T9 system reduces keystrokes at the expense of increased cognitive load (discussed extensively in Chapter 3).

Glyph improves on current systems by transparently learning user behavior and updating its internal model. It also uses a confidence parameter to interrupt the user with word predictions only when it the system is sufficiently confident that the prediction is correct. In addition, unlike the constantly changing display of T9 which can contain non-words, Glyph displays valid words when suggesting completions and valid prefixes as the user enters them. Errors do not ‘blow up’ (Section 5.7.4) unlike in T9. Character errors can be edited intuitively. It is also possible to freeze the prediction model of Glyph after the user is comfortable with the predictions. This will default the system to a form of optimized MultiTap with word completion.

However, Glyph is still plagued by the split-attention problem mentioned in Chapter 3. Even though there is a keystroke savings with the use of word prediction, there is not always an improvement in overall text generation or communication rate due to the costs of increased cognitive and perceptual loads [Klund et al, 2001].

7.6. Issues in Adaptation and Optimization

The automatic learning model overcomes the problem of mode switching in T9. Only a single mode of text entry is used for all actions in Glyph. T9 requires users to switch to MultiTap for words not found in the dictionary. However, this method has its drawbacks.

With explicit modeling, the user augments the system dictionary with personal words, albeit with extra effort. Glyph does not make any assumptions about the validity of the words that the users use in their messages. Because of this, misspelled words with even simple errors are susceptible to being added to the data model overloading it over time. One solution to this problem is to add words to the data model only if the words pass strict criteria like a minimum word size or a minimum frequency of occurrence. Another approach is to use algorithms like Levenstein distance to weed out misspelled words.

Another issue that can hinder adaptation to a user’s messaging style is the fact that it can take a significant number of messages before the user’s messaging statistics usurp the predefined statistics as the default prediction. This is true for both word and character prediction, but particularly severe for character predictions as the training corpus is typically huge compared to user written messages. A solution to this problem is to give special status to the information mined from the messages created by the user. A simple method of implementing this is to multiply the scores generated from user data by a modifier derived from empirical evaluations.

The data model can also perform frequent updates to its data by trimming entries that are rarely used or by reducing their scores. This also helps to hone the prediction model toward each user’s unique messaging style. In addition, the data model needs to be optimized further so that higher order n-grams can be utilized. Currently, the maximum n-gram order that can be loaded without optimization onto a phone is 3.

Figure 7.8. Predictions across the word boundary for order 3 and 4 n-grams

Bridging word boundaries was tested as a means to reduce the zero frequency problem. This has proven effective when higher order n-grams (~ 5) are used. The disk cost incurred with higher order n-grams have been discussed earlier. One solution to this problem is to selectively increase n-gram order for contexts that include a Space character as its penultimate element. The maximum ambiguity typically occurs for the first character of a new word. Hence, selective optimization of this nature is likely to achieve a good balance between accuracy and disk cost. Figure 7.8 shows the effect of n-gram order across word boundaries. A common pair of words is 'should be'. However, lower order n-grams do not have enough contextual information in their models to suggest 'b' after 'should'.

Chapter 8

Conclusion

8.1. Conclusion

8.1.1. Prototyping Framework

The prototyping framework is an object-oriented framework for text entry research on mobile devices. The architecture of the framework is founded on object-oriented design fundamentals. Case studies of current text entry methods were studied and the findings incorporated into the framework design. Two text entry methods were developed using the framework. The framework was found to be sufficiently adaptable to these tasks and there was a quantifiable decrease in development effort when using the framework.

Framework development is an iterative process. This has been stressed repeatedly in the report and in literature. The framework is still in its formative stages and has been used for prototyping Glyph. It will take many applications and continuous refinement in the design before the framework matures and its rigidities are identified and removed.

It is hoped that the framework will provide a boost to text entry researchers by helping them move from conceptualization to prototype with minimal effort.

8.1.2. Glyph

Glyph is an intelligent text entry system for mobile phones that transparently adapts to the user. The core of the system is a character-based predictor augmented with a small dictionary for added word-completion capabilities. This report has described the design and implementation of Glyph.

The report began with an overview of predictive text systems. A key contribution is the discussion on the nature of text messaging. Current systems that have influenced the design of Glyph were critiqued. The overall design of the predictive core was detailed and key design decisions taken in the design have been discussed along with the incurred tradeoffs. The mobile version of Glyph was evaluated based on established metrics and compared with current text entry systems.

8.2. Contributions of the Project

The first contribution of this project is the conceptualization, design and development of a prototyping framework for text entry research on mobile devices. An implementation of the framework won the *Runners up Award at the Encentuate Innovation Competition 2003*. A paper on the framework titled “A Prototyping Framework for Mobile Text Entry Research” was presented at the *6th Asian Pacific Conference on Computer Human Interaction (APCHI 2004)*. This report has also outlined the comprehensive literature survey and case studies performed as part of the project.

The major contribution of this project is *Glyph: An efficient and adaptive text entry system for mobile phones*. A paper on Glyph has been submitted to MobiCom 2005, the Annual International Conference on Mobile Computing and Networking.

The adaptive model of Glyph sidesteps the learning problem in current text entry systems by removing the need for text entry mode switches in learning thus avoiding the heavy cognitive loads associated with explicit learning models. In addition, several enhancements are proposed to enhance text entry rate. In tests, Glyph has been found to require *less than 1 key stroke per character*, significantly less than the 2 key strokes required for MultiTap and between the 1 and 1.6 key strokes required for T9. However, several tradeoffs (Chapter 6) have to be made to obtain impressive text entry rates, particularly in the size of the data model. In addition, cognitive effects and real world text entry rate can be evaluated only through tests with real users.

The report also contributes an analysis of the nature of mobile text messaging inferred from mining multiple data sources.

8.3 Recommendations for Future Work

A number of enhancements are proposed to improve the framework and Glyph.

8.3.1 Prototyping Framework

Integration of statistical methods

Many statistical models have been proposed for evaluation of text entry performance. These include the KSPC (Key Strokes Per Character) metric and Fitt’s Law [Fitts, 1954]. Integrating these methods into the framework and automating their tabulation is an important enhancement that can be made to the framework.

Support for auditing

Auditing user actions is an important requirement for usability testing. The open source Log4j logging system has been integrated into the framework and is expected to reduce the implementation effort for auditing.

8.3.2 Glyph

Improving Disambiguation

Glyph is currently primed using text that is not representative of informal conversation. The availability of a conversational corpus has been highlighted in Section 6.2. Priming the system with such a corpus is essential to provide quantitative statistics to prove the efficacy of the improvements incorporated into Glyph.

As was discussed in Chapter 7, character n-gram models provide diminishing returns for higher order n-grams. At this point, other approaches have to be implemented to improve text entry performance. The idea of augmenting character prediction by using n-gram word statistics has been briefly introduced in Chapter 2. This can be investigated further as a method to improve disambiguation accuracy.

Usability Evaluation and User Survey

An evaluation with real users is of paramount importance in studying real world text entry rates and the cognitive loads incurred. A well designed usability evaluation, with confounding factors properly controlled, is highly recommended to ascertain the usability and performance of Glyph. Better scientific models are also required to evaluate human factors issues in text entry.

Subjective satisfaction is critical for the success of any new text entry method. A user survey is also recommended to gauge user acceptance of the new text entry method.

Optimization

Mobile devices demand applications that are thrifty on computational and storage resources. A promising area of research is in optimizing the data model of Glyph so that higher order models can be loaded onto mobile devices and queried faster.

References

- Barber, 1997 Christopher Barber (1997). *Beyond the Desktop*. Academic Press, 24-28 Oval Road, London NW1 7DX, UK
- Booch, 1996 Grady Booch (1996). *Best Of Booch : Design Strategies for Object Technology*. SIGS Books and Multimedia 1996
- Bosch et al, 1999 Jan Bosch, Peter Molin, Michael Mattsson, PerOlof Bengtsson (1999). *Object-Oriented Frameworks—Problems & Experiences*. Department of Computer Science and Business Administration, University of Karlskrona/Ronneby
- Card et al, 1983 Card, S.K., Moran, T.P. & Newell, A. (1983). *The Psychology of Human-Computer Interaction*, Lawrence Erlbaum, New Jersey
- Chadenat et al, 2002 Olivier Chadenat and Thibaut Raballand (2002). *Predictive Text for Mobile Devices with Reduced Keyboards*
- Clear, 1993 Clear, J.H. (1993) 'The British National Corpus' in Paul Delany & G. P. Landow (eds) *The Digital Word : text-based computing in the humanities* . Cambridge, Mass.: MIT Press,: 163-187.
- Cook et al, 1995 A.M. Cook, S.M. Hussey, *Assistive Technologies: Principles and Practice*, Mosby-Year Book Inc., 1995, pp. 493-498.
- Copestake, 1997 A. Copestake (1997). *Augmented and alternative NLP techniques for augmentative and alternative communication. Proceedings of the ACL workshop on Natural Language Processing for Communication Aids*
<http://acl.ldc.upenn.edu/W/W97/W97-0506.pdf>
- Crystal, 2001 D. Crystal (2001). *Language and the Internet*. Cambridge University Press, Cambridge, UK, 2001.
- Darragh et al, 1990 J. J. Darragh, I. H. Witten and M. L. James (1990). *The Reactive Keyboard: A Predictive Typing Aid*. *IEEE Computer*, 1990. pp. 41-49.
- Davis, 1991 J. R. Davis (1991). *Let your fingers do the spelling: Disambiguating words spelled with the telephone keypad*. *AVIOS Journal*, 9:57–66, March 1991.
- Dunlop et al, 2000 A Mark Dunlop. *Text entry on mobile phones*. Risø National Laboratory, Denmark
- Dunlop et al, 2002 Mark Dunlop and Stephen Brewster (2002). *The Challenge of Mobile Devices for Human Computer Interaction*. Department of Computing and Information Systems, Department of Computing Science, University of Strathclyde, Glasgow, UK

- Dunlop et al, 2000 Mark D. Dunlop and Andrew Crossan (2000). Predictive text entry methods for mobile phones. *Personal Technologies*, 4(2):134-143
- Ellis, 2001 Eric Ellis (2001). Asia Buzz: Revolution:How text messaging toppled Joseph Estrada. *Time Magazine*
<http://www.time.com/time/asia/asiabuzz/2001/01/23/>
- Fait et al, 2002 Holly Auburn Fait and Jennae Bulat (2002). Word Prediction and Its Impact on Written Communication: Does the Use of Word Prediction Change How You Communicate?. Department of Computer Science, Berkeley, CA, 94720, USA May, 2002
- Fayad et al, 1999 Mohamed E. Fayad, Douglas C. Schmidt, Ralph E. Johnson. Building Application Frameworks : Object-Oriented Foundations of Framework Design. John Wiley & Sons 1999
- Fazly, 2002 Afsaneh Fazly (2002). The use of syntax in word completion utilities. *Master's Thesis*, Toronto University 2002
- Fitts, 1954 Fitts, P.M. (1954) The information capacity of the human motor system in controlling the amplitude of movement. *Journal of experimental Psychology*. 47, 381-381
- Foster et al, 2002 George Foster and Philippe Langlais and Guy Lapalme (2002). User-Friendly Text Prediction for Translators, RALI, Université de Montréal
- Fredkin, 1960 Edward Fredkin (1960). Trie Memory, *CACM*, 3(9):490-499
- Friedman et al, 2001 Zachary Friedman, Sarani Mukherji, Gary K. Roehm and Richesh Ruchir. Data Input Into Mobile phones: T9 or Keypad?. *SHORE 2001*,
<http://www.otal.umd.edu/SHORE2001/mobilePhone/index.html>
- Froehlich et al, 1997 Garry Froehlich, H. James Hoover, Ling Liu, Paul Sorensen (1997). Hooking into Object-Oriented Application Frameworks. Department of Computer Science, University of Alberta, Edmonton, AB. T6G 2H1
- Gamma et al, 1994 Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional Computing Series 1994
- Goldberg et al, 1983 Adele Goldberg, David Robson and Glenn Krasner (1983). Smalltalk-80: Blue Book: The Language and Its Implementation
- Grinter et al, 2003 R. Grinter and M. Eldridge (2003). Wan2tk?: Everyday text messaging. In *Proceedings of CHI 2003*, Fort Lauderdale, Florida, ACM press, New York, NY, 2003, 441-451.

- Hasselgren et al, 2003 Jon Hasselgren, Montnemery, Matthew Turk, Pierre Nugues and Markus Svensson (2003). HMS: A Predictive Text Entry Method Using Bigrams. *Proceedings of the Workshop on Language Modeling for Text Entry Methods, 10th Conference of the European Chapter of the Association of Computational Linguistics*
- Petrie, 1999 Petrie, Helen (1999) Writing in Cyberspace: A study of the uses, style and content of e-mail; <http://www.netinvestigations.net/>;
- Horstmann, 1997 Cay S.Horstmann (1997). Practical Object-Oriented Development in C++ and Java. John Wiley and Sons 1997 : Pgs 184, 121
- How, 2004 Yijue How (2004). Analysis of SMS Efficiency. Undergraduate Thesis. National University of Singapore.
- Hulme et al, 2001 Michael Hulme and Sue Peters (2001). Me, my phone and I: The role of the mobile phone. *Mobile Communications : Understanding Users, Adoption & Design, CHI 2002 Workshop*
- Hunnicutt, 1987 S. Hunnicutt (1987). Input and output alternatives in word prediction. *Technical Report* Ref: STL-QPSR 2-3, Department of Speech Communication and Music Acoustics, KTH, Stockholm, Sweden 1987.
- ISO, 1994 ISO/IEC 9995-8 (1994). Information technology - Keyboard layouts for text and office systems, Part 8: Allocation of letters to the keys of a numeric keypad.
- Isokoski, 1999 P. Isokoski (1999), A minimal device-independent text input method, *Master's thesis*, University of Tampere, Tampere, Finland.
- James et al, 2001 Christina L. James and Kelly M. Reischel (2001). Text input for mobile devices: comparing model prediction to actual performance. *Conference on Human Factors in Computing Systems* archive Proceedings of the SIGCHI conference on Human factors in computing systems Pages: 365 - 371
- Jurafsky, 2000 D. Jurafsky and J.H. Martin (2000), *Speech and Language Processing*
- JustText, 2003 JustText (2003). Text Facts <http://www.justtext.co.uk/sms.htm>
- Klund et al, 2001 J. Klund and M. Novak (2001). If Word Prediction Can Help, Which Program Do You Choose? <http://trace.wisc.edu/docs/wordprediction2001/index.htm>.
- Koester et al, 1996 H. H. Koester and S. Levine (1996). Effect of a word prediction feature on user performance. *Augmentative and Alternative Communication*, Volume 12
- Koester, Heidi Horstmann Koester (2002). Word Prediction - When does it

- 2002 enhance text entry rate? *RESNA 2002*, June 30, 2002
- Kolsch et al, 2002 Mathias Kölsch and Mathew Turk (2002). Keyboards without Keyboards: A Survey of Virtual Keyboards. Dept. of Computer Science, University of California at Santa Barbara, CA
- Lenhart et al, 2001 A. Lenhart, L. Rainie and O. Lewis (2001). Teenage life online: The rise of the instant-message generation. *Report of the Pew Internet & American Life Project*.
- Leshner et al, 2002 G. W. Leshner, B. J. Moulton, D. J. Higginbotham and B. Alsofrom (2002). Limits of human word prediction performance. *CSUN 2002*, California State University, Northridge.
- Leshner et al, 1999 Gregory W. Leshner, Bryan J. Moulton and D. Jeffery Higginbotham (1999), Effects Of N-gram Order And Training Text Size On Word Prediction. RI 1999 Lockport, NY: Enkidu Research
- Leshner et al, 1999 A Gregory W. Leshner and Gerard J. Rinkus (1999). Leveraging Word Prediction To Improve Character Prediction In A Scanning Configuration. Enkidu Research, Inc. Spencerport, NY 14559
- Leshner et al, 1998 Gregory W. Leshner, Bryan J. Moulton and D. Jeffery Higginbotham (1998). Techniques for augmenting scanning communication. *Augmentative and Alternative Communication*, 14, 81-101.
- Li et al, 2002 Yuanxiang Li, Xiaoqing Ding, Chew Lim Tan (2002). Combining Character-Based Bigrams with Word-Based Bigrams in Contextual Postprocessing for Chinese Script Recognition, *ACM Transactions on Asian Language Information Processing*, Vol. 1, No. 4, December 2002, Pages 297-309.
- MacKenzie et al, 2001 I. S. MacKenzie, H. Kober, T. Jones, and E. Skepner. Letterwise: Prefix-based disambiguation for mobile text input. In *Proceedings of the ACM UIST 2001*, pages 111 – 120. ACM, 2001.
- MacKenzie, 2002 I. S. MacKenzie (2002). KSPC (keystrokes per character) as a characteristic of text entry techniques. *Proceedings of the Fourth International Symposium on Human Computer Interaction with Mobile Devices*, pp. 195-210
- MacKenzie, 2002 A I. S. MacKenzie and R. W. Soukoreff (2002). Text entry for mobile computing: Models and methods, theory and practice. *Human-Computer Interaction*, 17, 147-198.
- MacKenzie, I. S. MacKenzie (2002). KSPC (keystrokes per character) as a

- 2002 B characteristic of text entry techniques. *Proceedings of the Fourth International Symposium on Human-Computer Interaction with Mobile Devices*, pp. 195-210. Heidelberg, Germany: Springer-Verlag.
- MacKenzie, 2003 I. S. MacKenzie and R. W. Soukoreff (2003). Phrase sets for evaluating text entry techniques. *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems – CHI 2003*, pp. 754-755 New York: ACM.
- Magnuson et al, 2002 Tina Magnuson and Sheri Hunnicutt (2002). Measuring the effectiveness of word prediction: The advantage of long-term use. Department of Speech, Music and Hearing, Stockholm, Sweden
- Mattson, 1996 Michael Mattson (1996). Object-Oriented Frameworks: a survey of methodological issues. *PhD Thesis*, University College of Karlskrona/Ronneby
- MobileYouth, 2002 MobileYouth (2002).
<http://www.mobileyouth.org/news/mobileyouth595.html>
- Purtilo, 1991 J. Purtilo, A. Larson and J. Clark (1991). A methodology for prototyping-in-the-large. *Proceedings of the 13th international conference on Software engineering*, Austin, Texas, pp. 2-12
- Rau et al, 1996 H. Rau and S. Skiena (1996). Dialing for documents: An experiment in information theory. *Journal of Visual Languages and Computing*, pp. 79-95, 1996.
- Rheingold, 2002 H. Rheingold (2002). *Smart Mobs: The Next Revolution*. Perseus Publishing, Cambridge, MA
- Saunders, 2003 Christopher Saunders (2003). Inching Toward Mobile IM February 28, 2003
<http://www.instantmessagingplanet.com/wireless/article.php/1963021>
- Shannon, 1950 Claude E Shannon (1950), Prediction and Entropy of Printed English, *Bell Systems Technical Journal* (3) p. 50-64.
- Shannon, 1948 C. Shannon (1948). A Mathematical Theory of Communication, *Bell Systems Technical Journal*, Vol. 27, pp. 398-403, 1948
- Shortis, 2000 Shortis, T. (2000). *The Language of ICT* London: Routledge (ISBN 0-415-22275-3). Corpus available at
<http://www.demo.inty.net/Units/SMS/corpus.htm>
- Silfverberg et al, 2000 M. Silfverberg, I. S. MacKenzie, and P. Korhonen (2000). Predicting text entry speed on mobile phones. *Proceedings of the ACM Conference on Human Factors in Computing Systems - CHI 2000*, pp. 9-16. New York: ACM.

- Sirisena, 2002 Amal Sirisena (2002). Mobile Text Entry. *Master's Thesis*, Department of Computer Science, University of Canterbury, Christchurch, New Zealand
- Soukoreff, 2003 R. W. Soukoreff and I. S. MacKenzie (2003). Input-based language modelling in the design of high performance text input techniques, *Proceedings of Graphics Interface 2003*, pp. 89-96.
- Straits Times, 2001 Straits Times (2001). SMS speaks Singlish here. 14 Jul 2001 http://it.asia1.com.sg/newsarchive/07/news006_20010714.html
- Stenstrom et al, 1993 Anna-Brita Stenstrom and Leiv Egil Breivik (1993). The Bergen Corpus of London Teenage Language (COLT) Department of English, University of Bergen, Sydnesplassen 7, N-5007 Bergen, Norway. <http://www.hit.uib.no/colt/>
- Tegic, 2003 Tegic Systems (2003). <http://www.t9.com/>
- Teich, 2001 Al Teich(2001). Technology and the Future Toolkit. 19 Mar 2001 <http://www.alteich.com/tidbits/t031901.htm>
- US Census, 2000 United States Census (2000). <http://www.census.gov>
- Wester, 2003 Malin Wester (2003). User Evaluation of a Word Prediction System. *Master's Thesis*. Uppsala University, Department of Linguistics, Language Engineering Program, 30th September 2003
- Werry, 1996 Werry, Christopher (1996) 'Linguistic and interactional features of Internet relay chat', in S.C. Herring (ed.), *Computer-Mediated Communication: Linguistic, social and cross-cultural perspectives*, Amsterdam:
- Yamada, 1908 H. Yamada (1908). A historical study of typewriters and typing methods: From the position of planning Japanese parallels. *Journal of Information Processing*, 2(4), 175-202.
- Zi Corp, 2003 ZiCorp (2003). <http://www.zicorp.com/ezitext.htm>
- 12Dicts, 2003 12Dicts compiled by Alan Beale available at <http://wordlist.sourceforge.net/12dicts-readme.html>

Appendix A

The Research Journal: Design and Implementation

1. Motivation

Research projects typically involve students performing extensive research, going through a huge number of publications, assimilating knowledge and preparing notes. The need was felt for a software application to automate these tasks and streamline the research workflow. A study of available solutions proved unsatisfactory to the needs at hand. Most research management software were either expensive or did not meet the personal needs of an individual researcher.

2. The Research Journal

The Research Journal is a web application developed specifically for the project to ease the procurement and management of research resources like notes, TODO lists, web sites and publications. It provides basic journal functions along with the ability to download and manage publications.

3. Design and Implementation

The Journal is implemented as a .NET web application using C#. The database is hosted on Microsoft SQL Server. The client interface was developed using DHMTL and MSHTML Rich Text Editing Engine. Publications are downloaded using the WebClient .NET component and catalogued in the database. Users can browse journal contents by date and view the linked documents using the multitabled interface.

Figure 1. Journal entries, TODO List and System Console

Figure 2. Rich Text Editor and Document Reference Manager

Figure 3. Integrated document viewer

Figure 4. Browse entries by date

Appendix B

Calculation of Degree of Disambiguity

There are two aspects to DoD calculation. We require a corpus of text to infer statistics from and a mobile phone design to apply these statistics to.

Given a corpus C , we can calculate the average frequency with which each alphabet occurs in C as $A_c, B_c, C_c \dots Z_c$ (several variations are possible if we consider alphanumeric and case sensitive frequencies). A generic formula for calculating the frequency of character N in corpus C is:

$$N_c = (\text{Number of times the alphabet } N \text{ occurs in the corpus } C) / (\text{Number of characters in } C)$$

We then split up the alphabets into different keys based on the design of the phone. For example, the standard mobile phone design assigns the number 2 key to characters (A,B,C) and the number 7 key to the set (P,Q,R,S).

DoD is then calculated for each key as the difference in frequency between the two highest frequency characters mapped to that key. For example, for key 7, out of the four characters mapped to the key, R and S are most frequent in the corpus C . Hence,

$$DoD7 = R_c - S_c$$

DoD is calculated for each key in a similar fashion.