UNIVERSITY OF THE WESTERN CAPE

The Use of Mobile Phones as Service-Delivery Devices in a Sign Language Machine Translation System

by

Mehrdad Ghaziasgar

A thesis submitted in fulfillment for the
degree of Master of Science

in the
Faculty of Science
Department of Computer Science

August 2010

Declaration of Authorship

I, Mehrdad Ghaziasgar, declare that this thesis titled, 'The Use of Mobile Phones as Service-Delivery Devices in a Sign Language Machine Translation System' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: ______________________________

Date: ______________________________

"All containers are reduced in capacity by what is placed in them, except a container of knowledge, which expands."

-Ali son of Abu Talib

UNIVERSITY OF THE WESTERN CAPE

Abstract

Faculty of Science
Department of Computer Science

Master of Science

by Mehrdad Ghaziasgar

This thesis investigates the use of mobile phones as service-delivery devices in a sign language machine translation system. Four sign language visualization methods were evaluated on mobile phones. Three of the methods were synthetic sign language visualization methods. Three factors were considered: the intelligibility of sign language, as rendered by the method; the power consumption; and the bandwidth usage associated with each method. The average intelligibility rate was 65%, with some methods achieving intelligibility rates of up to 92%. The average file size was 162 KB and, on average, the power consumption increased to 180% of the idle state, across all methods. This research forms part of the *Integration of Signed and Verbal Communication: South African Sign Language Recognition and Animation* (SASL) project at the University of the Western Cape and serves as an integration platform for the group's research. In order to perform this research a machine translation system that uses mobile phones as service-delivery devices was developed as well as a 3D Avatar for mobile phones. It was concluded that mobile phones are suitable service-delivery platforms for sign language machine translation systems.

Acknowledgements

The past 6 years have been a journey, the longest most difficult journey of my life. Countless challenges and ordeals. But I am here. I thank God for this blessing. For His help and for growing me from frailty into strength, from inability into competence and from fear into conviction.

I would to thank my parents. You toiled for me and supported me. You taught me so many things and this thesis has your essences in it. You believed in me and here I am. Thank you. I would like to thank Mr. Moses Kanaabi, my Grade 6 teacher. Sir, I don't know where you are, but this thesis has your essence in it. You believed in me when no one else did, not even myself. You found me and gave me direction and here I am. I would like to thank Mrs. Sharon Slinger. You believed in me and helped me excel and here I am. I would like to thank Dr. Roderick Julies. Sir, your positive outlook and your focus on transferring knowledge rather than discharging duty is praiseworthy. I enjoyed your lectures. I would like to thank the Rector of the University of the Western Cape Prof. Brian O'Connell. Sir, your positive, kind, humble, wise and helpful outlook is praiseworthy. Thank you for your help with the Imagine Cup. Thank you for everything.

I would especially like to thank Mr. James Connan. Sir, your expansive knowledge, wisdom, creativity, confidence and positive outlook never cease to leave me in awe. Under your supervision, I went from frailty into conviction. Your balanced approach to supervision is exemplary. You supported in times when it was needed most. I have always, do always and will always speak of you in acclaim. I am indebted to you always. Thank you.

I would like to thank all of the people that provided companionship during these six years. Afshin, Mohsen and Ayesha. Thank you. Fakhree, Theo, Riyaad and Badrudeen. Thank you.

Nathan, we went through some amazing experiences together. From the Imagine Cup South Africa awards ceremony, to doing four or five absolute last minute, but expertly done, presentations to putting together the first ever two-way sign language to English translation system. Thanks for everything buddy.

Jameel, we've been through some difficult times together. But here we are. Thank you for your friendship.

Contents

Declaration of Authorship **i**

Abstract **iii**

Acknowledgements **iv**

List of Figures **ix**

List of Tables **xi**

Abbreviations **xiii**

1 Introduction **1**

1.1 Background . . . 1

1.2 Research Objectives . . . 2

1.3 Research question . . . 3

1.4 Premises . . . 4

1.5 Thesis outline . . . 4

2 Communication Tools for the Deaf **6**

2.1 General-Purpose Communication Tools . . . 7

2.1.1 Text-Based Tools . . . 7

2.1.1.1 IM systems . . . 7

2.1.1.2 SMS . . . 8

2.1.1.3 Email . . . 8

2.1.1.4 Faxing . . . 8

2.1.1.5 Advantages and disadvantages of text-based tools . . . 9

2.1.2 Video-Based Tools . . . 9

2.1.2.1 Synchronous video communication tools . . . 9

2.1.2.2 Asynchronous video communication tools . . . 10

2.1.2.3 Advantages and disadvantages of video-based tools . . . 10

2.1.3 Bridging communication between devices . . . 11

2.2 Deaf-Specific Communication Tools . . . 13

2.2.1 Text-Based Tools . . . 13

2.2.2 Video-Based Tools . . . 14

2.2.3 Gesture-Based Tools 16
2.2.3.1 Spoken Language to SL Gesture-Based Tools 17
2.2.3.2 SL to Spoken Language Gesture-Based Tools 21
2.3 Summary 22

3 Java Micro Edition **24**
3.1 Java ME in Context – A Background on Java Editions 25
3.2 Java ME Architecture 25
3.2.1 The Java Community Process (JCP) 26
3.2.2 Configurations 28
3.2.2.1 The Connected Device Configuration (CDC) 29
3.2.2.2 The Connected, Limited Device Configuration (CLDC) . 30
3.2.3 Profiles 32
3.2.3.1 The CDC Profiles 32
3.2.3.2 The CLDC Profiles 33
3.2.3.3 The Mobile Information Device Profile 33
3.2.4 Additional APIs 35
3.2.4.1 The Mobile Media API 36
3.2.4.2 The Mobile 3D Graphics API 39
3.3 Summary and Conclusions 43

4 Sign Language Avatar Creation **44**
4.1 SL Avatar Creation 44
4.1.1 Avatar Modelling and Acquisition 45
4.1.1.1 Interactive Modelling 45
4.1.1.2 Parametric Modelling 46
4.1.1.3 Other Modelling Techniques 46
4.1.2 Avatar Parameterization and Deformation 48
4.1.2.1 Skeletal Subspace Deformation 48
4.1.2.2 Other Parameterization and Deformation Techniques . . 49
4.1.3 Avatar Animation 50
4.1.3.1 Keyframing 50
4.1.3.2 Motion Capture 51
4.1.4 Existing Standards and Tools 52
4.1.4.1 MakeHuman 52
4.1.4.2 H-Anim 52
4.1.4.3 MPEG-4 Facial Definition Parameters 53
4.2 Blender – an Open Source 3D Graphics Tool 54
4.2.1 Background 55
4.2.2 Tools and Features 56
4.3 Van Wyk's Methodology 57
4.3.1 Methodology Overview 57
4.3.2 Implementation of *Man* 58
4.3.3 Performance 60
4.4 Summary 62

5 Methodology and Implementation of the Avatars **63**

5.1 Creation of the Avatar *Phlank* . . . 64
5.1.1 Modelling and Acquisition . . . 64
5.1.2 Parameterization . . . 70
5.2 Animation and Exportation of *Phlank* and *Man* . . . 73
5.2.1 Animation . . . 73
5.2.2 Exportation . . . 74
5.3 Summary . . . 76

6 Implementation of a Prototype Mobile Framework 77
6.1 The SASL Machine Translation System . . . 78
6.1.1 Conceptual Overview . . . 78
6.1.2 Existing Components . . . 79
6.2 The iSign Mobile Framework Architecture . . . 80
6.3 Implementation of the iSign Mobile Framework . . . 81
6.3.1 The Communication Protocol . . . 82
6.3.2 The Web Service . . . 84
6.3.3 The Mobile Application . . . 85
6.3.3.1 The *Login* Screen . . . 85
6.3.3.2 The *Contacts* Screen . . . 86
6.3.3.3 The *Mode-Select* Screen . . . 87
6.3.3.4 The *Deaf Capture* and *Hearing Capture* Screens . . . 87
6.3.3.5 The *Word List* Screen . . . 89
6.3.3.6 Rendering Screens . . . 89
6.3.4 The Interpretation Services . . . 91
6.4 Summary . . . 91

7 Experimental Setup 92
7.1 Sign Language Intelligibility . . . 92
7.1.1 Collection of SASL Videos and Exportation of Sign Language Files 93
7.1.2 Viewing Sequence . . . 93
7.1.3 Experimental Setup . . . 94
7.2 Power Consumption . . . 96
7.3 Bandwidth Consumption . . . 98
7.4 Summary . . . 98

8 Results and Data Analysis 99
8.1 Sign Language Intelligibility . . . 99
8.2 Power Consumption . . . 107
8.3 Bandwidth Consumption . . . 109
8.4 Summary of Results and Discussion . . . 112
8.5 Summary . . . 113

9 Conclusion and Directions for Future Research 114
9.1 Feasibility Experimentation . . . 114
9.2 The iSign Mobile Framework . . . 115
9.3 Concluding Remarks . . . 115

A The “TestModel” Application **116**

B The iSign Communication Protocol **120**

B.1 Signals 120

B.2 Database Tables 121

Bibliography **124**

List of Figures

2.1 Four bridging scenarios, adapted from [45]. 12
2.2 A typical TDD device. 13
2.3 The Teldem [129]. 14
2.4 Halawani's architecture [46]. 20

3.1 Java editions and the devices they are built for. 25
3.2 The Java ME architecture. 26
3.3 The packages implemented by and the comparison between the CLDC 1.0, CLDC 1.1 and CDC 1.0. 30
3.4 The packages implemented by and the comparison between the MIDP 1.0, MIDP 2.0 and MIDP 3.0. 35
3.5 Graphical depiction of a simplified scene graph. 42

4.1 A segment of the H-Anim specification for the tree of bones, adapted from [3]. 54
4.2 MPEG-4 Facial Definition Parameter feature points [146]. 55
4.3 Overview of Van Wyk's Methodology, taken from [146]. 58
4.4 Completed *Man* model without (left) and with (right) a completed skeleton. 61
4.5 Completed *Man* model's face depicting MPEG-4 FDP facial feature points. 61

5.1 The arm model after extrusion of features up to and including the palm. . 65
5.2 A close-up of the glove strap segments. 65
5.3 The model and smoothed preview with Catmull-Clark subdivision surfacing level 1 applied. 66
5.4 A close-up of the model with background occlusion disabled to show five faces, A, B, C, D and E of the model. 67
5.5 (a) The resultant model and b) The resultant smoothed preview, after modelling the glove strap and palm. 68
5.6 The completed fingers. 69
5.7 The tip of the forearm (a) with the crease modified (b) with the crease unmodified. 69
5.8 The completed arms. 69
5.9 The completed *Phlank* model. 71
5.10 Hierarchical structure of *Phlank*'s skeleton. 73
5.11 The *Phlank* model with a completed skeleton. 73

6.1 Conceptual overview of the SASL Machine Translation system [28]. 78
6.2 Overview of the iSign Mobile Framework showing the process of a) SASL to English audio translation and b) English audio to SASL translation. . . 81

6.3 Sequence diagram of communication exchange signals. 84
6.4 The Sony Ericsson C905. 86
6.5 (a) The initial *Login* screen. (b) *Settings* screen for specifying the Translation Server URL. (c) Login details filled in. (d) Screen shown while sending information to the Translation Server and verifying login. 86
6.6 (a) The initial *Contacts* screen. (b) The *Contacts* screen with the 'Translator' entry selected. (c) Screen shown while sending information to the Translation Server and attempting to connect to the contact selected. . . 87
6.7 The *Mode-Select* screen for Translator mode. 87
6.8 (a) The initial *Deaf Capture* screen. (b) The video viewfinder. (c) The video viewfinder in the process of recording. (d) The video viewfinder after recording is complete. (e) The *Deaf Capture* screen with a recording loaded. (f) The screen displayed while the file is being sent. 88
6.9 (a) The initial *Hearing Capture* screen. (b) The audio capture window. (c) The audio capture window in the process of recording. (d) The audio capture window after recording is complete. (e) The *Hearing Capture* screen with a recording loaded. (f) The screen displayed while the file is being sent. 89
6.10 The *Word List* screen. 90
6.11 (a) The user is prompted whether or not to play the message retrieved. (b) The Avatar *Phlank* incorporated into iSign. (c) The message log depicting four messages. 90

7.1 The SASL word 'Sick' as rendered by a) the low-detail Avatar *Phlank* (LowRes) b) the Avatar *Man* with non-manual gestures (Facial) c) the Avatar *Man* without non-manual gestures (NoFacial) and d) sign language video (SLVid). 93
7.2 Experimental design used to determine power consumption. 97

8.1 Percentage of correctly identified phrases per test subject. 100
8.2 Percentage of correctly identified phrases per method. 101
8.3 Percentage of correctly identified viewings per phrase. 103
8.4 Number of correct identifications per method for the four phrases with recognition rates of less than 50%. 104
8.5 Number of correct identifications per method for the phrases in which SLVid and Facial performed better than NoFacial and LowRes. 105
8.6 Frequency of reasons given for incorrect identification of phrases. 105
8.7 Percentage increase in power consumption by each method. 110
8.8 File size of each phrase for each SL visualization method. 111
8.9 Average file size of each method. 112

List of Tables

3.1 A list of key JSRs . . . 29
3.2 Player states, their descriptions and the methods used to invoke them. The states occur sequentially starting at Unrealized and ending at Closed. 38
3.3 Standard Controls in the MMAPI . . . 38
3.4 Key classes in the M3G API. . . . 40
3.5 Key methods provided by the Graphics3D instance. . . . 42

4.1 Sample list of H-Anim-assigned names for bones . . . 53
4.2 MakeHuman parameter values for the creation of *Man*, adapted from [146] 59
4.3 Rotational limits placed on specific bones, adapted from [146] . . . 60
4.4 Geometry details of *Man*, adapted from [146] . . . 60

5.1 Number of BUs by which each segment was extruded. . . . 64
5.2 Number of BUs by which segments in each finger were extruded. . . . 67
5.3 Rotations and shifts applied to fingers. . . . 68
5.4 Features removed from *Phlank* in ascending order of importance until the model was importable and renderable on the mobile phone. . . . 71
5.5 The total number of vertices, edges and faces on the completed *Phlank* model. . . . 71
5.6 The bones in the right arm, their assigned names, and the position of their tips relative to their parent bone. . . . 72
5.7 List of SASL phrases recorded and animated. . . . 74
5.8 Key parameters of the M3G exporter tool. . . . 75
5.9 Features removed from *Man* in ascending order of importance. . . . 75

6.1 Key components of the iSign Mobile Framework. . . . 82
6.2 Signals in the communication protocol and their descriptions. . . . 83
6.3 Description of the function of each table in the iSign web service database. 85

7.1 Arrangement of method-word viewing groups. . . . 94
7.2 Example viewing sequences of subjects 1 and 2. . . . 94
7.3 Test subjects used in the intelligibility experiment. . . . 95

8.1 Correctly identified phrases per test subject. . . . 100
8.2 Number of viewings required before an attempt at recognition was made. 101
8.3 Correctly identified viewings per method. . . . 101
8.4 Differences of Least Squares Means for pair-wise comparisons between the four methods. . . . 102
8.5 Correctly identified viewings per phrase. . . . 102

8.6 Updated number of correctly identified viewings per method after removal of problematic words. . . . 106
8.7 Average amount of current drawn $\bar{I}$ for each phrase of each method and the standard deviation in the mean s. . . . 108
8.8 Average amount of current drawn $\bar{I}$ by each method and the standard deviation in the mean s. . . . 108
8.9 Z-test result for each pair-wise comparison. . . . 109
8.10 Increase in current drawn $\bar{I}$, standard deviation s and the percentage increase in current drawn. . . . 109
8.11 File size for each phrase of each SL visualization method. . . . 110
8.12 Average file size of each method and the percentage standard deviation from the mean. . . . 111
8.13 Summary of the performance of each SL visualization method in terms of the factors considered. . . . 113

B.1 Signal pertaining to *Login* on the phone. . . . 120
B.2 Signals pertaining to the *Connect* on the phone. . . . 120
B.3 Signals pertaining to the *Job Upload* on the phone. . . . 121
B.4 Signals pertaining to the *Job Processing* on the server. . . . 121
B.5 Signal pertaining to the *Job Retrieval* on the phone. . . . 122
B.6 Data dictionary of the USER table. . . . 122
B.7 Data dictionary of the CONTACTS table. . . . 122
B.8 Data dictionary of the VIDEOIN table. . . . 123
B.9 Data dictionary of the WAVEIN table. . . . 123

Abbreviations

3D	Three Dimensional
2D	Two Dimensional
API	Application Programming Interface
ArSL	Arabic Sign Language
ASL	American Sign Language
Auslan SL	Australian Sign Language
AWT	Abstract Windowing Toolkit
BSL	British Sign Language
BU	Blender Unit
CDC	Connected Device Configuration
CLDC	Connected, Limited Device Configuration
CSL	Chinese Sign Language
CVM	C Virtual Machine
DOF	Degrees of freedom
DSTBP	Digital Set Top Box Profile
EC	Executive Committee
EU	European Union
FPS	Frames Per Second
GR	Gesture Recognition
GUI	Graphical User Interface
HMM	Hidden Markov Model
IM	Instant Messaging
IMP	Information Module Profile
I/O	Input/Output
ISL	Irish Sign Language

Java ME	Java Mobile Edition
Java SE	Java Standard Edition
JCP	Java Community Process
JSR	Java Specification Request
JVM	Java Virtual Machine
KSL	Korean Sign Language
KVM	K Virtual Machine
M3G	Mobile 3D Graphics
MB	Megabyte
MIDP	Mobile Information Device Profile
MMS	Multimedia Messaging Service
MT	Machine Translation
NURBS	Non-Uniform Rational B-Splines
OpenGL	Open Graphics Library
OpenGL ES	OpenGL for Embedded Systems
PC	Personal Computer
PSTN	Public Switched Telephone Network
RAM	Random Access Memory
ROI	Region Of Interest
ROM	Read-Only Memory
SASL	South African Sign Language
SASL Project	Integration of Signed and Verbal Communication: South African Sign Language Recognition and Animation Project
SASL Group	Integration of Signed and Verbal Communication: South African Sign Language Recognition and Animation Group
SATSA	Security and Trust Services API for J2ME
SBML	Systems Biology Markup Language
SDK	Software Development Kit
SL	Sign Language
SMS	Short Messaging Service
SVM	Support Vector Machine
SWML	Sign Writing Markup Language
TCK	Technology Compatibility Kit

TDD	Telecommunications Device for the Deaf
TISSA	Telephone Interpreting Service for South Africa
TSL	Taiwanese Sign Language
TTY	Telephone Typewriter
UI	User Interface
UK	United Kingdom
US	United States
URL	Uniform Resource Locator
UWC	University of the Western Cape
VH	Virtual Human
VM	Virtual Machine
VRML	Virtual Reality Markup Language
VRS	Video Relay Service
X3D	Extensible 3D

Chapter 1

Introduction

1.1 Background

There are over one million deaf people in South Africa [44]. Approximately 300,000 of these are profoundly deaf in both ears and use South African Sign Language (SASL) as their first and only language [44]. Contrary to common belief, sign languages are not visual-gestural representations of spoken languages. They are rich fully-fledged languages of their own [117]. Additionally, different countries have sign languages of their own. Examples include Greek Sign Language (GSL) in Greece, Japanese Sign Language (JSL) in Japan, Arabic Sign Language (ArSL) in Saudi Arabia, British Sign Language (BSL) in the United Kingdom and South African Sign Language (SASL) in South Africa. Each of these sign languages also experience regional variations within the country. This is akin to dialect and idiolect variations in spoken languages [117].

The majority of deaf people in South Africa are illiterate in spoken languages [44]. Even those deaf people that have been trained in spoken languages are not able to read and write at the same proficiency level as hearing people [29] [116].

These facts have contributed to the creation of a firm communication barrier between the deaf and hearing [40]. Dolnick remarks in [30] that Helen Keller said, "Blindness cuts people off from things. Deafness cuts people off from people". Even so, deaf people necessarily need to communicate with hearing people on a daily basis. Research has shown that 90% of deaf children are born to hearing parents. These children need to communicate with their hearing parents on a daily basis as a necessity of life. Other situations include accessing dental and medical services, purchasing bus and train tickets, seeking directions and purchasing groceries. The inability to access these services effectively because of communication constraints is a source of stress for the deaf [47].

Another significant problem arising from this communication barrier is that deaf people face discrimination in employment opportunities [19]. There is a reluctance to employ people that are illiterate and cannot communicate or be communicated with. This has lead to a state of poverty amongst the South African deaf [44].

The solution hitherto employed to remedy this situation was the use of highly skilled and trained interpreters [146]. The use of interpreters has proven inadequate and inefficient. There is an acute shortage of such interpreters [44] [146]. Their services are also very costly [146]. Most deaf people in South Africa cannot afford this service. Also, the use of interpreters is unsuitable in contexts in which confidentiality is vital such as when a deaf person seeks medical or psychological treatment. In such cases, the deaf person may not be keen to have a human interpreter present.

The *Integration of Signed and Verbal Communication: South African Sign Language Recognition and Animation* project (henceforth referred to as the SASL project) at the University of the Western Cape [28], of which this research is part, has proposed a Machine Translation (MT) system that will eventually be able to carry out automated translation between SASL and English. Ideally, given video input, the system aims to produce and render the corresponding English audio, and given English audio as input, the system aims to produce and render the corresponding sign language video. Milestones have been achieved in this project with the production of sign language recognition [88] [106] [144] [99] and rendering [146] systems. Naidoo [88] and Rajah [99] produced gesture recognition systems that can track hand-motions, Segers [106] produced a hand-shape recognition system and Whitehill [144] produced a robust facial animation recognition and classification system. Van Wyk [146] produced a full-body high-resolution humanoid Avatar that can be used to produce synthetic sign language phrases in the form of animations. However, these systems were not unified and existed as independent systems built to run on desktop PCs.

It is a key aim of the SASL project to make the services of such a translation system accessible on mobile phones. Prior to this study, the feasibility of implementing such a system had not yet been investigated. Several feasibility factors needed to be considered and investigated in order to inform the development process of the SASL project and other projects involved with translation between sign language and spoken language.

1.2 Research Objectives

First and foremost, it is necessary to determine whether or not mobile phones have the required application programming interfaces (APIs) for the implementation of such a

MT system. Mobile phones would be required to record audio and video, communicate with a server over the Internet and play back audio and video. We intend to investigate this.

Secondly, it is necessary to investigate the feasibility factors of displaying synthetic sign language on mobile phones. To our knowledge, no study has dealt with this issue comprehensively. It is crucial to establish whether or not synthetic sign language displayed on a mobile phone screen is intelligible to deaf people. This is, by far, the most important feasibility factor. It is also desirable to investigate the bandwidth and power consumption associated with displaying synthetic sign language on a mobile phone. These factors are of less significance but are, nevertheless, useful in informing projects dealing with the display of synthetic sign language on mobile phones.

This research focuses on pioneering this much-needed feasibility study and furthering the expanse of knowledge in this field in a bid to inform the SASL project and other projects in the field of translation between sign language and spoken language.

As an extra step, we desired to apply our feasibility findings and show that an implementation of the SASL project's Machine Translation system, that uses mobile phones as translation service-delivery devices, is feasible, by designing and implementing a working proof-of-concept version of it. To our knowledge, no unified Machine Translation systems exist, either on desktop PCs or on mobile phones, that provide both sign language to spoken language **and** spoken language to sign language translation. As such, this implementation would constitute a pioneer in the field of translation between sign and spoken languages as well as constituting a highly significant milestone in the SASL project.

1.3 Research question

Are mobile phones feasible service-delivery devices in a sign language Machine Translation (MT) system?

This question is broken down into the following sub-questions:

1. Do mobile phones have the required APIs for the service-delivery needs of the sign language MT system?
2. Is synthetic sign language rendered on a mobile phone intelligible to deaf people?
3. What are the power and bandwidth consumption implications of displaying sign language on mobile phones?

4. Can we design and implement a working proof-of-concept sign language MT system that uses mobile phones as a service-delivery mechanism?

1.4 Premises

The past decade has witnessed trends of rapid development and cost reduction in mobile device technology. Thus, mobile devices considered novel just two years ago have now been replaced by far superior technologies and the high cost of those devices has reduced significantly. For this reason, the SASL project makes use of the latest technology in its implementations in order to remain as up-to-date as possible.

Based on this premise, this research adopts the latest most suitable mobile device technology at the time of its commencement. We assume, based on this reasonable premise, that towards the completion of the SASL project, deaf people will be able to obtain mobile devices that have comparable capabilities to the one we select and use.

1.5 Thesis outline

The rest of this thesis is organized as follows:

- **Chapter 2** *Communication Tools for the Deaf*: This chapter presents a survey of existing tools that can be used to translate between sign and spoken language. These are dealt with based on those that were (deaf-specific tools) and were not (general-purpose tools) specifically created to facilitate communication between the deaf and the hearing. We mention the use of general-purpose tools, such as instant messaging systems and the short messaging service (SMS), and any research studies conducted to assess the feasibility factors of these tools to the problem of deaf-hearing communication. We also present a survey of research projects aimed at creating tools to solve the problem of deaf-hearing communication and the studies, if any, that have aimed to assess the feasibility factors of these tools.

- **Chapter 3** *Java Micro Edition*: This chapter evaluates the suitability of the Java Micro Edition (Java ME) programming language in providing the required APIs for service-delivery functions in a MT system. A top-down overview of Java ME is provided, specifically focusing on and explaining those components that satisfy the feasibility requirements under investigation. We also mention the mobile phone that provided these required components that we selected for use in our feasibility study.

- **Chapter 4** *Sign Language Avatar Creation*: In this chapter, we explain the generic process used to create sign language Avatars that can be used to display synthetic sign language. This is the process that we used to create a sign language Avatar and Van Wyk used to create his Avatar[146], both of which we used in our feasibility study. We explain the tool Blender which we used to carry out the generic Avatar creation process and explain its suitability to our needs.

- **Chapter 5** *Methodology and Implementation of the Avatars*: This chapter explains the implementation of the Avatars *Man* and *Phlank*. We explain the creation and parameterization of *Phlank*, as well as the animation and exportation of both *Man* and *Phlank*. This is in accordance with the generic Avatar creation process explained in Chapter 4.

- **Chapter 6** *Implementation of a Prototype Mobile Framework*: In this chapter, we provide details of our implementation of a working proof-of-concept sign language MT system that uses mobile phones as a service-delivery mechanism, as per our objectives. We provide a structural overview of the framework and explain our implementation of each component in the framework.

- **Chapter 7** *Experimental Setup*: This chapter explains the experimental design we used to test the three feasibility requirements: intelligibility; power consumption; and bandwidth usage.

- **Chapter 8** *Results and Data Analysis*: The results of the experiments carried out are summarized in this chapter. An analysis of these results is carried out and explained.

- **Chapter 9** *Conclusion*: This is the final chapter of the thesis and draws conclusions from the analysis of the results explained in Chapter 8. It also provides directions for future work in this field.

Chapter 2

Communication Tools for the Deaf

In this chapter we describe existing tools that can facilitate deaf-hearing communication. As is expected, these tools are communication tools that employ a non-auditory mode of communication suitable for deaf use. Tools that do not fit into this category are excluded. As will be explored, and as noted in [148], deaf people have a limited set of communication tools available to them.

These tools can be subdivided primarily according to their generality of use. This yields: those tools that are built for general-purpose communication and are suitable for deaf-hearing communication as well; and those tools that have been built specifically to facilitate deaf-hearing communication. The next two sections describe these tools in detail. Where possible, we mention any studies done that relate to the effectiveness of a particular tool in solving the problem of deaf-hearing communication. The chapter concludes with a summary.

It is important, at this point, to mention that deaf people can be subdivided into two distinct groups; literate and illiterate[1] deaf people. The second group comprises those deaf people whose first and only language is Sign Language (SL). They have no knowledge of spoken languages. They are, for all practical purposes, completely unable to communicate with hearing people. This group is referred to as the deaf with a capital letter *D* – the Deaf [98] [69]. The first group comprises those deaf people that have been educated in spoken languages and are able to lip-read, speak and/or read and write written text. For this reason this group is able to communicate with the hearing in spoken languages with varying degrees of proficiency. This group is referred to as the deaf

[1]Traditional definition of literacy – literacy as applied to proficiency in reading and writing in spoken languages.

with a small letter *d* – the deaf [98] [69]. With regards to interaction with the hearing world, the literate group are in a much better position than the illiterate group. They are less marginalized from the hearing community and suffer from fewer communication limitations. As mentioned in the previous chapter, the large majority of deaf people in the South African context are illiterate [116]. This is taken into consideration when evaluating the effectiveness of the tools mentioned within the South African context.

2.1 General-Purpose Communication Tools

General-purpose communication tools are those that have been developed for communication in general but are also suitable for deaf-hearing communication. We subdivide these tools according to the modality of communication exchange, into two categories. Each category is discussed in each of the following subsections: text-based tools; and video-based tools. As will be explained, some of these tools require additional infrastructure and human-intervention in order to facilitate deaf-hearing communication. In other words, communication needs to be bridged between the deaf and hearing. The subsection that follows describes how bridging has been achieved.

2.1.1 Text-Based Tools

The general-purpose text-based tools that can be used for deaf communication include, but are not limited to: general-purpose Instant Messaging (IM) systems; the Short Messaging Service (SMS); Email; and Fax. Power *et al.* [98], Bowe [19], Pilling *et al.* [96] and Power *et al.* [97] conducted studies on the use of these tools by deaf people in Australia, the United States (US), the United Kingdom (UK) and Germany, respectively. A comparison of these studies reveals that the preference of applying a particular tool to a particular communicative task varies sharply from region to region and depends heavily on social and cultural norms. Where possible, we indicate the preferences of each text-based tool in each of the regions mentioned.

2.1.1.1 IM systems

IM systems are systems that allow for synchronous real-time text-based communication over a network which, in the general case, is the Internet. They can be accessed on a variety of platforms that include mobile phones and personal computers (PCs). Examples of such systems include MXit and Fring-Mini which are mobile-based systems, and PC-based systems such as Yahoo! Messenger, MSN Messenger and GTalk. IM systems are the closest text-based alternative to a telephone call. This is because they are synchronous in nature, highly responsive and incorporate a presence-service [38]. On the other hand, they are dependent on both communicating parties being present in the same time span, and do not incorporate dialling facilities. Bowe [19] suggests that this communication tool is very popular amongst American deaf people.

2.1.1.2 SMS

SMS allows for asynchronous or semi-synchronous text-based communication. The service is mainly accessed on mobile phones but it is also possible to send SMSs from a PC to mobile phone over the Internet. In cases where there are long periods of time between messages, it is considered asynchronous in nature, but if the communicating parties send messages in rapid succession, the communication may be reclassified as semi-synchronous. According to Power *et al.* [97], this communication tool is the most popular tool amongst deaf people in Germany. Also, Power *et al.* [98] state that, given a choice of only a single text-based communication tool, the majority of deaf people in the UK would prefer this communication tool. Pilling *et al.* [96] also state that this communication tool is the most popular amongst deaf youth in the UK. It is mostly used for personal- and social-related matters [98] but is not suitable for more formal communication requirements [44].

2.1.1.3 Email

Email, like SMS, may be asynchronous or semi-synchronous, depending on the nature of communication taking place. Emails can be sent and received on PCs and most modern day mobile phones. Power *et al.* [98] suggest that this mode of communication is preferable for formal and business-related communication purposes, since the previous two tools are generally unsuitable for such purposes. Bowe [19] and Pilling *et al.* [96] indicate that this form of communication is popular in the US and UK, respectively.

2.1.1.4 Faxing

Faxing is, by-and-large, asynchronous in nature. Faxes can be sent by means of facsimile devices as well as PCs. They are similar to email in terms of the format of the message content and, as suggested by Power *et al.* [98], are preferable for personal and business-related interactions.

2.1.1.5 Advantages and disadvantages of text-based tools

Text-based tools are relatively low cost, which is favourable to deaf people. A very significant advantage of such systems is the fact that they enable the deaf involved to communicate with the hearing, without the need for any additional intermediary tools. Also, the fact that they are ubiquitous makes them accessible. According to Pilling *et al.* [96], they are preferable because they are fast and easy to use. For this reason, they are popular among the deaf, albeit in the developed world [98] [19] [96] [97].

This popularity, however, owes itself to the fact that the deaf people involved are able to read and write text. The ability to read and write is critical to using such tools as noted by Bowe [19]. Unfortunately, this is a minority case in the South African context since, as mentioned in Chapter 1, the majority of South African deaf people are illiterate [44] [116] and have a low aptitude for reading and writing English, even if trained [116] [146]. In addition, the fact that conventional deaf communication makes use of signing, and such tools do not, has raised concerns that such tools threaten to “wipe out” Deaf culture entirely [98]. Therefore, such tools cannot serve as a general solution to the problem of deaf communication in South Africa. Other pros and cons of this mode of communication exist that apply to all user groups, deaf and non-deaf alike, such as the difficulty incurred in expressing a variety of emotions [140] [102]. These factors are not discussed.

2.1.2 Video-Based Tools

Video-based tools carry out a video message exchange. The deaf person can record SL using a webcam. This video is then sent over the network to another person who views it. Such tools require additional translation infrastructure in order for the deaf person to communicate with a hearing person. These are discussed in Section 2.1.3. The general-purpose video-based tools available today can be broadly sub-divided into two categories based on the synchronicity of communication employed, namely: synchronous video communication tools; and asynchronous video communication tools.

2.1.2.1 Synchronous video communication tools

Synchronous video communication tools allow all communicating parties to see and hear each other in real-time. Cameras and microphones capture the images and voices of communicating parties which are then sent over a network, either the PSTN or the Internet. Numerous software clients exist for PCs that facilitate synchronous video communication including Skype, Google Talk and Yahoo! messenger. All of these operate over the Internet. Modern mobile phones provide video calling and video conferencing capabilities which operate over the cellular network. Also, certain specialized tools such as Videophones and Video conferencing equipment allow for video calling.

2.1.2.2 Asynchronous video communication tools

Asynchronous video communication tools involve the sending of video over the network, but with significant periods of time between messages from communicating parties. Two significant examples of such tools are video mail and the Multimedia Messaging Service (MMS). Video mail is a video version of email. Similar to email, a video message is captured and sent to a recipient's video mail box, and can be accessed by the recipient at a later time. The video mail service is divided into those services that require a client to be downloaded to the sender's PC, to be used for capturing and sending purposes, and those that provide a web-based capture and send mechanism, with no additional software required. Prominent providers of video mail of the former type are Comcast [27] and Sightspeed [108], and prominent providers of the latter type are Eyejot [37] and TokBox [132]. The Multimedia Messaging Service, similar to the email/video-mail analogy, is a video version of SMS. The mobile phone camera is used to capture a short video, which is then sent as a message to another mobile phone user.

2.1.2.3 Advantages and disadvantages of video-based tools

Video-based communication is the most natural form of SL communication after face-to-face communication [69] since the deaf communicating party is able to speak in SL. Their cultural norms can be expressed and they are not required to be literate in spoken languages. Emotions can also be communicated with great ease by the use of facial expressions.

However, such systems do not allow deaf people to communicate with hearing people on an as-is basis, and require additional intermediary systems to be in place, better known as bridging systems. Bridging systems are dealt with in the next subsection, Section 2.1.3. Therefore, the ability of these tools to solve the problem of deaf-hearing communication depends on the availability of bridging tools in a particular region. As is noted in the section that follows, Section 2.1.3, no video bridging tools are available in South Africa. This rules out the use of these tools as a solution to the problem of deaf-hearing communication in South Africa. Other less significant challenges also exist. Research has shown that deaf people become concerned with their appearance when using video-based communication unlike in the case of text-based communication. This is because the appearance of the communicating parties needs to be presentable and cannot be used, say, in the middle of the night out of bed [69]. Also, it has been found that this type of communication is not completely natural to SL speakers. Given the limited visual field of webcams [69], certain signs cannot be signed naturally. For example, when signing signs that required the hands to be moved to the space to the far sides of the person, it is necessary for the signer to move away from the webcam to capture the whole gesture in the camera's field of view [69]. However, the desperate need for these services means that these challenges can be overlooked.

2.1.3 Bridging communication between devices

When using devices of the same modality, communication between the two parties is direct. No intermediary system is required. However, when devices of different modality are used by two communicating parties, such as a text-based instant messenger by one party and a sign-language video-based video-chat application by the other party, it becomes necessary to put in place a system that effects the necessary translation between the two modalities. Such systems can either be machine- or human-based.

They are referred to as relay services. Relay services effect many communication modality switches such as text to speech and vice versa, SL video to speech and vice versa, etc. Theoretically, relay services should be able to switch between any two communication modalities, devices and applications. However, this is not the case. Most relay services currently in place worldwide switch between text and speech and between video and speech.

Thus, a relay service that converts between text and speech must be able to convert text to speech and speech back into text, and a relay service that converts between video and speech must be able to convert SL video to speech and speech back into SL video. Ideally, all such conversions should be carried out in an automated fashion by a machine in order to make the process efficient, effective and low-cost. As is discussed in Section 2.2.3, this has not yet been fully achieved. Therefore, most current relay services are human-based.

Typically, a deaf person, by means of a text-based platform such as a text-based IM on a computer or a video-based platform such as a Videophone, calls a relay service operator with the same device in front of them. The operator is, in most cases, a highly trained SL interpreter. The deaf person requests a connection to a phone number. The operator then dials the number on a telephone and makes contact with the other party. In this way, the operator is connected to both parties simultaneously on the different devices. Thereafter, a turn-taking method is used in which the communicating parties send alternating messages to the operator who translates it and sends the message across to the other device. Figure 2.1 has been adapted from [45] and depicts several such scenarios.

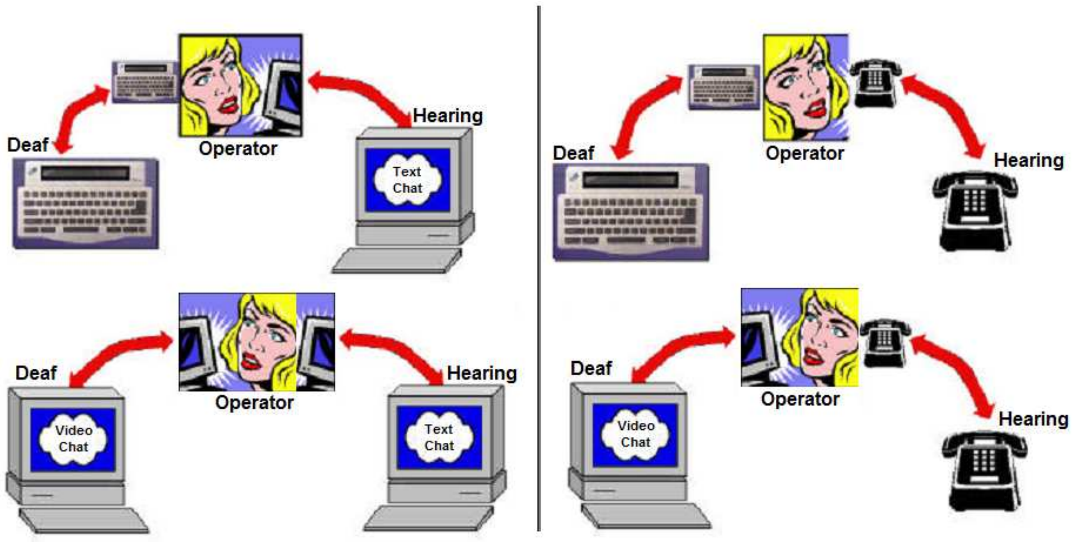


FIGURE 2.1: Four bridging scenarios, adapted from [45].

Examples of relay services include the Royal National Institute for the Deaf's TypeTalk of the United Kingdom [134], the Australian Communication Exchange's National Relay Service [10], Australian-based Talking Text [127] and American-based AT&T's TTY relay service [9]. No national relay service is available in South Africa [44]. This renders video-based communication tools unable to solve the problem of deaf-hearing communication in South Africa. Attempts, however, have been made at providing bridging services of limited capability in South Africa.

TalkingSMS is a service provided by the cellular telecommunications company Vodacom in South Africa. It converts SMS text messages sent by mobile phones into speech played back on landline telephones and is seen as a valuable tool for facilitating communication between the deaf and hearing [138] [44]. Another project that provided bridging services was the Telephone Interpreting Service for South Africa (TISSA) [128] that was piloted in 2002. This project incorporated all eleven South African official languages, as well as SASL. Two communicating parties in the same location, and only in specific locations such as government offices with special infrastructure in place, could dial in to an interpreter in a remote location. Using a videophone for SASL and telephone for spoken language, the interpreter would then bridge communication between the two parties. This bridging service was, however, distinct from a relay service since both participants had to be in the same location to communicate, and only specific locations were catered for.

2.2 Deaf-Specific Communication Tools

Deaf-specific communication tools are those that have been developed specifically to facilitate communication between the deaf and the hearing. We subdivide them, again, according to the modality of communication exchange. We again yield the following two subsections: text-based tools; and video-based tools. A third type of tool is introduced in this section that makes use of Machine Translation (MT) to translate between English and SL. We refer to this type of tool as a gesture-based tool since these tools exchange and use SL gesture information.

2.2.1 Text-Based Tools

Telecommunications Devices for the Deaf (TDD), based on the Telephone Typewriter (TTY), are the main text-based type of devices that are designed specifically for the deaf. They typically take the form of a small laptop-sized device with a keyboard and a display screen or, in older devices, a printer, and have a connection to the Packet Switched Telephone Network (PSTN) [69] [45]. Figure 2.2 depicts a typical TDD device. The operator of the device types text on the keyboard of the device. The text is converted into tones which are then sent over the PSTN using Baudot encoding. Finally, the tones are converted back into text by the TDD on the other end of the call and displayed on the screen or printed to the printer. The South African version of a TDD is the Teldem, developed by the South African-based telecommunications company Telkom. Figure 2.3 depicts the Teldem. Like most other TDDs, it communicates in half-duplex mode, meaning that only one party is able to type at a given time. As such, a method of turn-taking between the communicating parties is employed while communication is taking place.

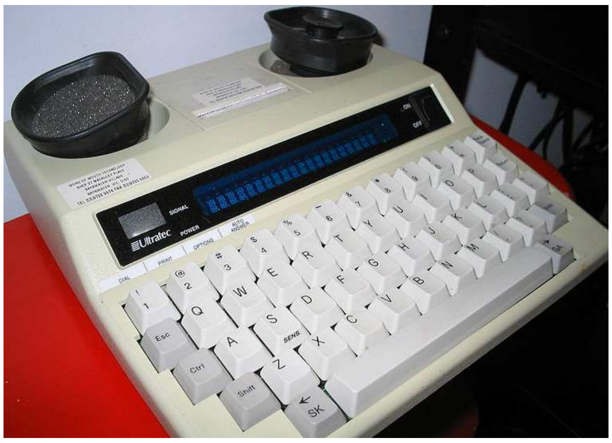


FIGURE 2.2: A typical TDD device.

FIGURE 2.3: The Teldem [129].

Having dialling capabilities and employing a synchronous mode of communication, these devices are the closest deaf alternative to telephones. As mentioned, however, they are half-duplex, making them unlike traditional SL communication [69]. Many modifications to TDDs have been proposed including, but not limited to: simulated full-duplex capabilities by an alternate transmission of packets [32]; increasing the character set and the speed of data transfer by means of a backward-compatible yet improved communication protocol [33] [35] [34]; and increasing the typing efficiency by introducing an augmentation device that helps predict the word being typed based on a network based dictionary service [101].

A substantial number of deaf people in developed countries use this tool [98] [96]. This is due to the fact that the deaf of those regions are able to read and write text, required to be able to use this tool, which is a minority case in the South African context, and the fact that those regions have relay services. As has been mentioned, this is not the case in South Africa and the Teldem is not a common tool amongst deaf people in South Africa [133]. Like other text-based tools, it is an ineffective solution to the problem of Deaf communication in South Africa.

Other text-based tools have also been developed specifically for the deaf but are similar to one or more of the general-purpose tools. The tool described in [6], for example, is very similar to mobile phone-based instant messaging systems with minor modifications. These are not discussed further.

2.2.2 Video-Based Tools

Attempts at creating video-based tools specifically for use by deaf people have, by-and-large, taken the form of minimizing the limitations of the general-purpose video-based communication tools. Specifically, attempts have targeted one or both of the following two areas with a much greater emphasis on the first, namely: reduction of transfer cost and time of the video over the network i.e. reduction of video size; and increase in the usability of the application specifically for SL. Thus, research has mainly aimed at taking general-purpose video-based communication tools and improving them to make them suitable for deaf communication.

At the forefront of research in the reduction of video size is the optimization of codecs – algorithms used to compress videos by removing redundant information. When selecting and/or optimizing a codec for use with SL video, a balance needs to be struck between the quality of the video, the quantity of the data needed to represent the compressed video, the complexity of the encoding and decoding algorithms which will proportionally increase the time required for both operations, robustness of the compression scheme to data losses and errors, and a number of other factors. Research has shown that the H.264 encoding standard provides the best such balance possible at the time of writing [78]. As such, most projects involving video compression, including those involving SL video, have used codecs based on this encoding scheme [23] [85] [105] [86].

When optimizing codecs for SL video, a popular technique has been to assess key regions of interest (ROIs) in SL videos and to optimize codecs to provide better quality in those regions, while reducing the unnecessary quality in other regions of the video. Projects such as the MobileASL project [23], as well as those of Muir *et al.* [85] [86] and Seersb *et al.* [105] have all carried out ROI assessments for SL videos by tracking the gaze of SL-speakers viewing SL videos. Their research has consistently and unanimously shown that the key region of interest is the facial region, specifically the region around the mouth and chin, with the gaze of viewers fixed on this region for the greatest portion of the viewing time. As such, these projects also optimized their codecs, based on the H.264 standard, to provide a higher quality in the facial region and lower quality in all other regions of the video. Also, the MobileASL project was able to determine that providing fewer higher quality frames per second had a much higher utility for SL video than providing a higher number of lower quality frames per second [23].

The usability of video-based communication tools have also been the subject of research. The HIT-wear project [103] involved the use of a wearable computer mounted on the head of a deaf user, equipped with a screen and camera. System events, such as selecting menus, were initiated by pointing the index finger of the one hand at one of five fingertips of the other hand, each of which represented one of five available menus. Other projects, such as that of Zhu *et al.* [149], attempted to provide hand-driven interfaces in a similar manner. Although these systems aim at providing a more familiar interface to deaf users, since they enable the use of hands to carry out various functions, the requirement of a wearable computer poses as an excluding factor since wearable computers are by no means ubiquitous. This is especially not the case amongst the South African deaf.

While we have presented these enhancements to video-based tools for reasons of completeness, we re-iterate that video-based tools do not solve the problem of deaf-hearing communication in South Africa for reasons mentioned in previous sections.

2.2.3 Gesture-Based Tools

Gesture-based tools refer to all those tools that are built specifically to translate between SL and spoken language. They use Machine Translation (MT) to effect this translation. As noted previously, all other types of tools either do not require translation, such as text-based tools, or use human translators to carry out translation, such as video-based tools. We refer to this type of tool as a gesture-based tool since these tools generate and/or use SL gesture information. We elaborate on this shortly. These tools are conceptually required to cater for two distinct translation pipelines:

1. Translation from spoken language to SL.
2. Translation from SL to spoken language.

In translating from SL to spoken language, gesture recognition (GR) algorithms are used to recognize SL from some form of SL input, such as a SL video or wired gloves[2] fitted on a person speaking SL. The output of these systems is information pertaining to the gesture(s) made in the SL input. This may be in a SL notation system such as SignWriting [125], Stokoe [117], HamNoSys [135] [136] [137] or simple gloss notation. This output is then analyzed and translated into a spoken language.

Conversely, in translating from spoken language to SL, the input which is some form of spoken language, audio or text, is translated into SL gesture information. Again, this could be in one of the SL notation systems mentioned previously. This SL gesture information is then used to synthesize SL. The most prominent methods used to synthesize SL are the use of a 3D humanoid Avatar and SL video blending/joining.

The fact that these systems use SL gesture information sets them apart from other types of tools. We term these gesture-based tools.

For a system to be able to provide two-way communication between deaf and hearing users, both translation pipelines would have to be included in the same system. We

[2]Wired gloves are a set of gloves fitted with sensors to gauge motion or position. Examples are Data Gloves, CyberGloves, Power Gloves and AcceleGloves.

are not aware of any systems that incorporate both translation pipelines. All of these systems have aimed at implementing one of the translation pipelines mentioned. For this reason, we treat the projects done on each translation pipeline separately in each of the two following subsections. Also, most of these systems have been implemented on desktop computers, with very few implemented on mobile architectures. In each subsection that follows, we first mention desktop computer-based systems followed by those implemented on mobile architectures, where applicable.

2.2.3.1 Spoken Language to SL Gesture-Based Tools

The translation from spoken language to SL involves the following processes: capturing spoken language input as audio or text; if audio has been captured, converting it into text using speech recognition technology; translating the text into a SL transcription notation such as those mentioned previously; and finally synthesizing SL using the SL transcription. Various projects have carried out variants and subsets of this process. Some have attempted to implement as much of it as possible while others have focused on perfecting particular parts of the process.

Capturing spoken language is trivial and can be carried out by means of a microphone. Also, many speech recognition technologies currently exist that can convert from spoken language audio to text, many of them open-source such as Simon [110], CMU Sphinx [26] and VoxForge [139]. On the other hand, translation from spoken language text to a SL transcription notation and SL synthesis from that notation are the subject of current SL research. The majority of research on this translation pipeline focus on these two processes.

Additionally, some of these projects have carried out assessments on the intelligibility of their SL, while others have not. We will indicate those that have.

Two projects that are actively working on and have achieved milestones in such systems for SASL are the SASL project [28] of the University of the Western Cape and the SASL-MT project [150] of the University of Stellenbosch. The SASL project has, to-date, focused on SL synthesis and, more recently, SL synthesis from a SL transcription notation. Van Wyk [146] created a full-body anatomically-correct humanoid Avatar that can sign SASL gestures, including all non-manual gestures. His Avatar's model was built on the MakeHuman project [79]. Its skeleton was built on the H-Anim standard and he extended this standard to include additional features and bones for full facial animation. We explain his work in more detail in Chapter 4.

The SASL-MT project has focused both on translating from English text to SL transcription notation, in this case the gloss notation, and on synthesizing SASL. Van Zijl and Olivirin [152] of this group produced a graphical user interface (GUI) that allows a hearing person to compose English sentences for translation into SASL. The resulting sentence is then translated into SASL using a tree-based rule-based translation. It is not clear how robust and extensive their system is in this regard. Van Zijl and Fourie [151] of the same group developed a generic 3D humanoid Avatar that is capable of gesturing manual signs. Their Avatar was built using the H-Anim standard which is the standard in humanoid Avatar skeletons. In this way, it conforms to standards and is skinnable. They also developed a custom SL transcription format called SignStep. SignStep is an XML representation of the SL notation and includes details of the particular gesture to be animated, such as hand position, motion and shape. Their Avatar could be fed gestures in the SignStep format. It would then animate the appropriate sign. Their Avatar, however, is only capable of animating a small number of facial expressions. This is because the H-Anim standard, which they used, only incorporates a small number of animation specifications in the face. Plans are in way, by the project, to implement facial animations [151].

Many projects also exist that attempt to translate from spoken languages into non-SASL SLs. The most prominent of these projects was the ViSiCAST project [70] [29] that attempted to translate from English to British Sign Language (BSL) and other European SLs. The project aimed at implementing a full translation system for English to BSL in post offices in the UK. They used a commercial-off-the-shelf speech recognition package that was very robust. Their system could also parse the resulting text and translate it into a computer-readable Extensible Markup Language (XML) form of HamNoSys called SiGML. The SiGML would then be parsed by a module called Animgen to determine the sequence of BSL gestures required [70]. Their Avatar, Tessa, then gestured the resulting BSL animation. Tessa could perform body and hand animations but could not perform facial animations. The whole project is closed-source and very few details are available as to exact implementations. The system could recognize and animate 115 distinct BSL phrases that are used in day-to-day post office interactions [29].

This project is particularly applicable to our research in that it is one of the few projects that we know of that has carried out SL intelligibility evaluation of their Avatar with deaf people [29], albeit on desktop computers and not mobile devices. We mention their experimentation methodology. Their experimentation was conducted in a post office environment. From their 115 phrases, they generated 133 phrases by incorporating numbers and days of the week. These phrases were made up of a total of 444 signs, with repetitions of signs between different phrases. For example, the sign for "Pound" may have been used in multiple phrases involving money [29]. Using these signs, they conducted two types of experiments, the first of which was objective and the second, subjective. We focus on the first experiment that objectively tested intelligibility of the signs.

Six deaf people took part in the experimentation. Each deaf person was shown all 133 phrases in blocks of between 20 or 24 phrases per viewing, with gaps in between viewings. For each phrase, the deaf person was instructed to write down what they understood. The deaf person was able to control the playback of the phrase and could repeat the animation until it was intelligible or a maximum of 5 viewings had occurred without the sign having been intelligible. The results were evaluated by determining the percentage of signs in each phrase that had been identified correctly as well as determining the percentage of signs that had been identified correctly for each sign across all phrases. On average, 61% of phrases and 81% of signs were identified correctly. It was found that, on average, 1.8 viewings were required before an attempt at recognition was made. Each incorrectly identified phrase was then re-presented along with its text to the deaf person. The deaf person was asked to specify whether the error made was attributed to the SL phrase being inappropriate, such as in a different dialect or accent of BSL, or just not clear. It was found that 30% of errors made were due to inappropriateness of the phrase and the remaining 70% was attributed to unclear signing.

Morrissey [83] created a system similar in function to the previous one for English to Irish Sign Language (ISL) translation. Her system takes in English text phrases and uses an existing system to translate the English into ISL in the form of gloss-notation. Her system then displays the corresponding ISL by means of a dictionary look-up of pre-stored videos of animations of her Avatar for the required signs. She also carried out experiments with deaf people, the details of which are unclear. She achieved an 82% intelligibility rate for her ISL signs.

Many other projects have implemented Avatar systems that can animate signs on desktop computers from a computer-readable SL notation similar to the projects mentioned, with variation at the implementation level. These include those done for Greek to Greek Sign Language [68], Korean to Korean Sign Language (KSL) [89], English to Australian Sign Language [145] [147] and Polish to Polish Sign Language [41], to mention but a few prominent examples. These vary in terms of the technologies their Avatars are built on and the SL notation system they use to control and animate them. Some of them can perform non-manual gestures [89] [145] [147] while others can only perform manual gestures [68] [41]. The general concept remains the same. No studies that we are aware of have been carried out to assess the intelligibility of any of these systems.

Other projects have implemented translation systems on mobile devices, albeit for non-SASL SLs. Halawani [46], for instance, proposed a mobile architecture for the translation of Arabic text into Arabic Sign Language (ArSL). His architecture consists, mainly, of a mobile device, a proxy server and a web server. The mobile device sends text to the web server for translation, apparently through the web browser on the mobile device. The web server's translation module performs a translation of the text into ArSL and sends back the corresponding ArSL animation to the mobile device. Few details are provided beyond this. His overall architecture resembles the one proposed in the current research. His architecture is illustrated in Figure 2.4.

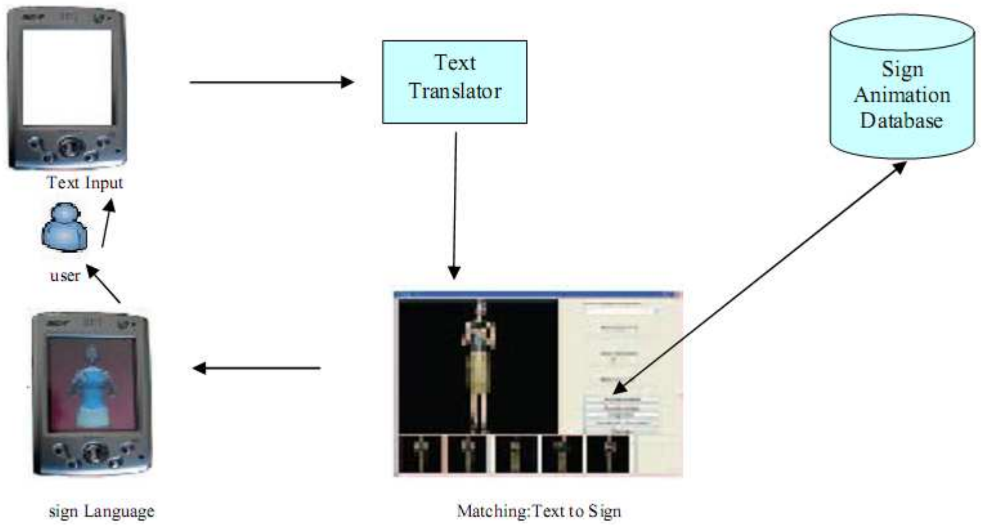


FIGURE 2.4: Halawani's architecture [46].

Igi *et al.* [51] developed a similar system for mobile devices that could animate manual gestures as well as 19 representative facial expressions for Japanese Sign Language (JSL). Their architecture, however, is limited in scope and size, making use of a limited-radius infrared wireless system. It was implemented in a museum. It consists of a PDA (Personal Digital Assistant) hosting Windows Mobile, a position-detecting server and an information-providing server. The position-detecting server is used to track the position of the PDA in the limited-size network using an infrared sensor and transmitter on the server and PDA, respectively. The information-providing server can be queried and sends back a video of a JSL-signing Avatar. Their system had a dictionary size of not less than 60 words, although the exact size is not clear. Their SL animations were created by means of specialized and expensive motion-capture equipment and led to high quality signing. They also carried out intelligibility evaluations on their Avatar with deaf people. Seven deaf people were shown 60 JSL words using their Avatar, 3 times each. They achieved an overall average recognition rate of 77% for their signs across all participants.

Chittaro *et al.* [25] [21] developed a system that allows novice-animators to animate Sign Language signs with relative ease and subsequently play those files on certain mobile devices. Their animation tool HAnimator allows novice users to create SL poses by a method of GUI-based rotations of joints on their Avatar. The user is also able to piece poses together to create SL animations. These animations may then be exported in the X3D format [2]. The X3D format is a 3D vector graphics format for 3D objects. They also developed an X3D file player for mobile devices called MobiX3D [82]. This allows for the animation files previously created to be played back. It was compatible with Windows Mobile PDAs. No studies were carried out to assess the effectiveness of the approach in rendering intelligible SL.

As is observed, very little work has been done to provide mobile device-based MT tools for deaf-hearing communication. Also, although work has been done to assess the feasibility of such systems, none of it has focused on the South African situation and SASL.

2.2.3.2 SL to Spoken Language Gesture-Based Tools

The translation from SL to spoken language involves the following processes: capturing SL input; carrying out gesture recognition (GR) on the input to extract SL information; carrying out translation from SL gesture information to spoken language in text; using text-to-speech technology to convert from spoken language text to spoken language audio; and finally rendering the audio. Different projects have carried out variants and subsets of this process.

Capturing SL video input is a trivial process that can be carried out by a video camera, motion-detection equipment or wired gloves. Text-to-speech technology is a highly researched topic and many robust text-to-speech systems exist, many of which are open-source such as Festival [39], FreeTTS [42] and eSpeak [36]. On the other hand, GR for SL is a relatively new field [115] [84].

GR systems can be classified broadly according to: those that make use of data from obtrusive equipment such as Data Gloves [76], Power Gloves [67], CyberGloves [141], AcceleGloves [49], coloured gloves [20] and motion-detection equipment as input; and those that use video data from a single camera [114] [7] as input known as vision-based systems. The output of GR systems may include manual gestures, which are hand movements and shapes, and non-manual gestures which include facial expressions and head, neck, shoulder and body movements. The output of GR is then used with a variety of techniques to classify the SL sign according to that information, including the use of correlation, Hidden Markov Models (HMMs), Neural Networks and decision trees. Below, we briefly mention work done in the field, first mentioning the projects dealing with SASL followed by those that deal with other SLs. We do not focus on the details of GR or translation techniques since this is not the subject our research.

The only project actively working on such systems for SASL is the SASL project [28] of the University of the Western Cape. With regards to this particular translation pipeline, the SASL project has mainly produced systems that can carry out GR. It has further focused on making these GR systems vision-based systems. Rajah [99] and Naidoo [88] both produced vision-based systems that can recognize SASL gestures. Both their systems made use of HMMs to classify signs. Rajah's system could recognize 23 SASL signs and achieved a recognition accuracy of 71%. Naidoo's system could classify 20 SASL signs and achieved a recognition accuracy of 73%. Whitehill [144] created a vision-based system that uses Support Vector Machines (SVMs) to classify and recognize SASL facial expressions from real-time input with an accuracy of 90%. Also, Segers [106] created a vision-based hand-shape recognition system from real-time input. His system uses eigenvectors to classify and recognize SASL hand shapes with an accuracy of 74%. The SASL project has not focused on translation from SL gesture information to English at all. To our knowledge, no other systems currently exist that carry out GR for SASL signs and no systems exist at all that carry out translation from SASL GR output into English.

Many projects have also attempted to translate from other SLs into spoken languages. The most prominent of these projects are those that have done so for American Sign Language (ASL) to English [114] [20] [49], Arabic Sign Language (ArSL) into Arabic [7], Taiwanese Sign Language (TSL) into Taiwanese [76], Australian Sign Language (Auslan SL) into English [67] and Chinese Sign Language (CSL) into Chinese [141].

These projects have achieved varied recognition accuracies that range from 73% [20] to 97% [114]. They also use varying techniques to classify and translate signs such as correlation analysis, HMMs and decision trees, and can recognize a varied number of signs ranging from 30 [7] to 5100 [141] signs. Some of these projects use obtrusive hardware [20] [49] [76] [67] [141] while others use vision-based techniques [114] [7]. Vision-based techniques are preferred over the use of obtrusive hardware since the latter is usually specialized, expensive and/or undesirable as it adds additional and unrealistic hardware constraints to the system, while the former does not. None of the systems mentioned cater for SASL.

2.3 Summary

We have presented a summary of the tools that may be considered capable of addressing the problem of deaf-hearing communication. We subdivided them, primarily, according to the generality of use and then according to the modality of communication employed and exchanged by the tools. We showed that the majority of the existing tools, general or specific to the deaf, are not effective in the South African context. We also showed that the work done on developing tools specifically for the deaf in South Africa and SASL is very limited and demonstrated the pressing need for the current research.

Chapter 3

Java Micro Edition

Our mobile phone application, iSign Mobile, was developed using Java Micro Edition (Java ME). In this chapter we describe the Java ME programming language used to create applications for Java-enabled mobile devices. We describe its structure and the features that it provides for rich application functionality. In so doing, we demonstrate its suitability to our needs. Developing a proof-of-concept SASL Machine Translation system using mobile phones as service-delivery mechanisms was stated as an objective in Chapter 1. From a usability perspective, such a system would require that the mobile phone application used as a service-delivery mechanism have the following capabilities:

1. Ability to capture video and audio.
2. Ability to render video, 3D graphics and audio.
3. Ability to communicate with a server over the internet.

As such, this chapter investigates whether Java ME provides the required APIs to provide these basic capabilities.

This chapter first gives a background of Java ME, distinguishing it from other Java versions. It then explains the process by which Java ME APIs develop in the Java Community Process. It also describes the process by which these APIs are structured and standardized into three specification types. These are: Configurations such as the Connected Device Configuration and the Connected, Limited Device Configuration; Profiles such as the Mobile Information Device profile; and Optional APIs such as the Mobile Media API and the Mobile 3D Graphics API. Each of these is explained in detail indicating those that are used in our implementation.

3.1 Java ME in Context – A Background on Java Editions

Sun Microsystems has developed Java in a variety of editions, catering for a range of target platform capabilities. These editions are as follows: Standard, Enterprise, Micro and Java Card. Common-place Java, as found on most desktop computers, is the Standard edition of Java – Java 2 Standard Edition (Java SE). It includes a set of core Java libraries as well as the Java Virtual Machine (JVM). The JVM mediates between the Java program and the underlying hardware on the platform so as to enable a "write-once-run-anywhere" approach [119]. It allows for an abstraction of the underlying hardware from the programmer to enable a standardized programming approach. A full specification of the JVM is available on the official Java website [90]. Java SE allows for the development of software for desktop-computers with highly sophisticated capabilities and Graphical User Interfaces (GUIs). The Enterprise Edition of Java – Java EE – was built on top of Java SE, with additions made to cater for server-side development.

Java SE, while highly portable, was too big to be implemented on devices with limited memory and processing capabilities. At the same time, many interfaces that would be required and useful on mobile devices but not on desktop computers were not provided by Java SE. An example of such an interface is the JSR177 package – the Security and Trust Services API (SATSA) which is used to provide a security verification mechanism for applications developed for mobile phones. As such, the need arose for compact versions of Java SE that could run on devices with more limited capabilities. The two remaining editions of Java – Micro and Java Card editions – catered for this need. Java Card was developed to run on smart cards and Java Micro Edition was developed for devices whose hardware capabilities fall in between that of desktop computers and smart cards [143]. Figure 3.1 depicts the various Java editions and the devices they are built for. This chapter focuses specifically on Java ME, excluding all other editions of Java.

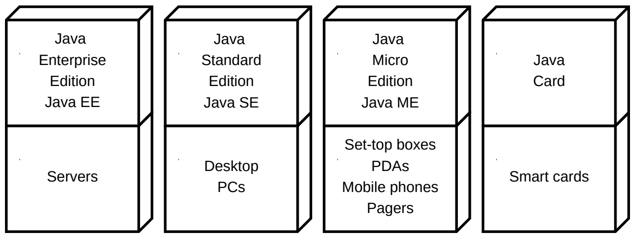

Java Enterprise Edition Java EE
Servers
Java Standard Edition Java SE
Desktop PCs
Java Micro Edition Java ME
Set-top boxes PDAs Mobile phones Pagers
Java Card
Smart cards

FIGURE 3.1: Java editions and the devices they are built for.

3.2 Java ME Architecture

Java ME is not a programming language on its own [93]. Rather, it is an adaptation of other existing Java technologies to make them suitable for devices less powerful than desktop computers [93]. The Java ME architecture is comprised of three components that are stacked on top of each other as follows; configurations, profiles and optional APIs. A configuration specifies a fixed subset of both the JVM [90] and the Java SE APIs that should be implemented by a given device to be able to run basic Java applications. A profile specifies the libraries available for the development of more sophisticated applications on a specific family of devices. The additional APIs allow access to specialized capabilities on a particular device. Each component in this layered architecture, therefore, provides an increasing level of specialized hardware access on devices they are aimed at. Each component is explained in further detail in the subsections that follow. The combination of a configuration, a profile and the optional APIs that a device has implemented is referred to as a stack [73]. The different configurations, profiles and optional API stacks are illustrated in Figure 3.2.

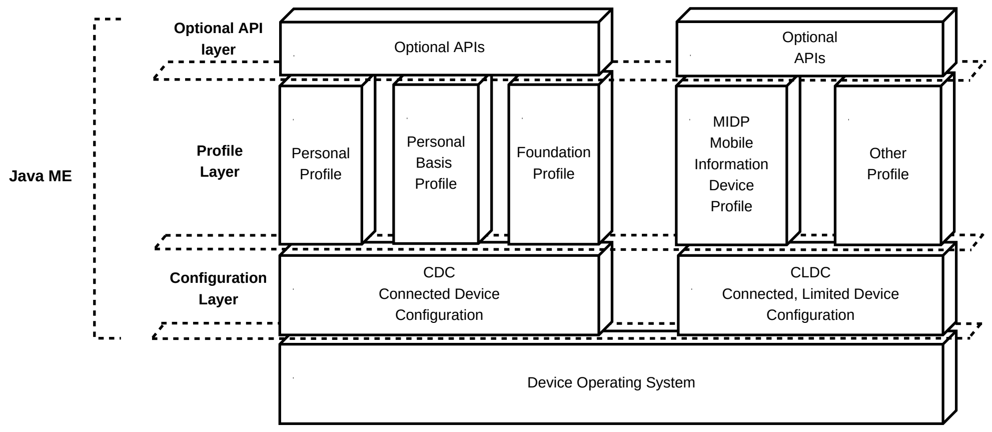


FIGURE 3.2: The Java ME architecture.

Different devices can be constructed to implement various configurations and profiles, each implementing varied interfaces and APIs. This form of construction would cause a severe lack of standardization which would make development for these devices very difficult. For this reason, a finite set of standard configurations and profiles have been developed. Device manufacturers must then select and conform to one of these configurations and profiles when constructing a particular mobile device. Section 3.2.1 explains the Java Community Process (JCP) and how Java ME components are developed, in order for the reader to fully understand the components of a Java ME stack explained thereafter. The subsections that follow then describe the components of a Java ME stack.

3.2.1 The Java Community Process (JCP)

The Java Community Process (JCP) was established in 1998. The role of this process was to provide a mechanism by which different Java technologies could be developed by consensus [57]. Anyone can register with the JCP and become a member[55]. Any member can propose or request a specification of a Java technology. These specifications are termed as Java Specification Requests (JSRs). JSRs can serve many functions including the following [73]:

- Proposal of new API(s) that should be integrated into Java technology to cater for new hardware.
- Extension of an existing JSR. Thus, a new JSR can be an extension to an existing JSR, specifying additional features that the old JSR should implement.
- Amendment of an existing JSR. A new JSR may propose solutions to the drawback(s) of an existing JSR.
- Combination of multiple other JSRs into functional units. For example, a JSR may be proposed to combine the JSR 82 (Java APIs for Bluetooth) [66] with the JSR 135 (Java Mobile Media API) [58] if all devices in use were capable of supporting both of those JSRs.

JSRs may serve one or more of these functions. For example, a JSR may be proposed to, both, amend and extend an existing JSR. The current development process of JSRs takes place as follows [57]:

1. **Initiation**: A member of the JCP proposes a specification. This is then briefly reviewed by an Executive Committee (EC) composed of major stakeholders in that technology, such as representatives of manufacturing companies, as well as members of the JCP.
2. **Early draft**: A group of experts is established consisting of members of the JCP. This group develops a preliminary draft of the specification. This is then put on the Internet where members of the JCP and the general public can review and comment on it, the comments received are used by the group to re-assess and improve the specification.
3. **Public draft**: The specification is put up for review by the general public again. The new feedback is used by the expert group to further amend the specification. At this stage, the EC decides whether or not the specification should continue. If accepted, a reference implementation of the specification and a Technology Compatibility Kit (TCK) are completed and provided to the EC for final approval by the leader of the expert group. The TCK is a set of tools and tests that verify the validity of a particular implementation of the specification. This shall be further explained shortly.

4. **Maintenance**: The completed specification and its reference implementation and TCK are amended and extended on an ongoing basis to cater for demands. Requests for clarification and interpretation are also catered for.

A JSR is only a specification. No working programs are produced or provided under the JCP. It is the responsibility of device manufacturers to produce an actual implementation of a JSR on a device based on the reference implementation of the JSR [73]. For this reason, implementations of JSRs may differ slightly or greatly, usually depending on the manufacturer of the device. TCKs provide a mechanism of verifying a particular manufacturer's implementation of the specification for accuracy and correctness. Nevertheless, implementations between manufacturers do vary.

The JCP itself has been the subject of several JSRs [56]. In June of 2000, the original JCP was replaced by the JCP 2.0. The JCP 2.0 made amendments to the process of making new JSR submissions and was proposed by the JSR 913 [54]. Maintenance on this JSR led to refinements on the rules for voting and resulted in the introduction of the JCP 2.1 in July of 2001. Further maintenance on this JSR then led to a revision of the rules of licensing and changes in policy which produced the JCP 2.5 in October of 2002. A new JSR – the JSR 215 [53] – then made further changes to the JCP, amending mistakes in voting, amending the format of the TCK, Reference Implementation and Specification, proposing more transparency and proposing changes in the EC structures, to name but a few. This produced the JCP 2.6 in May of 2006. The current version of the JCP is the JCP 2.7, produced by maintenance done on the JSR 215.

Table 3.1 summarizes key JSRs mentioned in this chapter and others used in our implementation.

3.2.2 Configurations

A configuration is a specification of a particular JVM and a set of core APIs that will be used to run and construct Java applications. It provides very basic functionality such as (but not necessarily limited to) the support for Java data types, input/output (I/O) operations including network connections and the provision of fundamental data structures such as Hash tables and Stacks.

Different configurations accommodate varying levels of standard mobile device processing and memory capabilities. A configuration is, therefore, a specification of those components of Java SE that can be run on a given class of mobile device. It categorizes mobile devices at the lowest level and is generally described in terms of the minimum memory capacity that it requires a mobile device to have. Currently, two standard

TABLE 3.1: A list of key JSRs

Type	JSR Name	JSR No.
Configuration	Connected Device Configuration CDC 1.0	36
	Connected, Limited Device Configuration CLDC 1.0	30
	Connected, Limited Device Configuration CLDC 1.1	139
Profile	Digital Set-Top Box Profile DSTBP	242
	Foundation Profile	46
	Information Module Profile IMP	195
	Mobile Information Device Profile 1.0	37
	Mobile Information Device Profile 2.0	118
	Mobile Information Device Profile 3.0	271
	Personal Basis Profile	129
	Personal Profile	62
Process	Java Community Process 2.0, 2.1 and 2.5	913
	Java Community Process 2.6 and 2.7	215
Optional API	Java APIs for Bluetooth	82
	Location API for Java ME	179
	Mobile 3D Graphics M3G	184
	Mobile Media API MMAPI	135
	Wireless Messaging API WMA	120

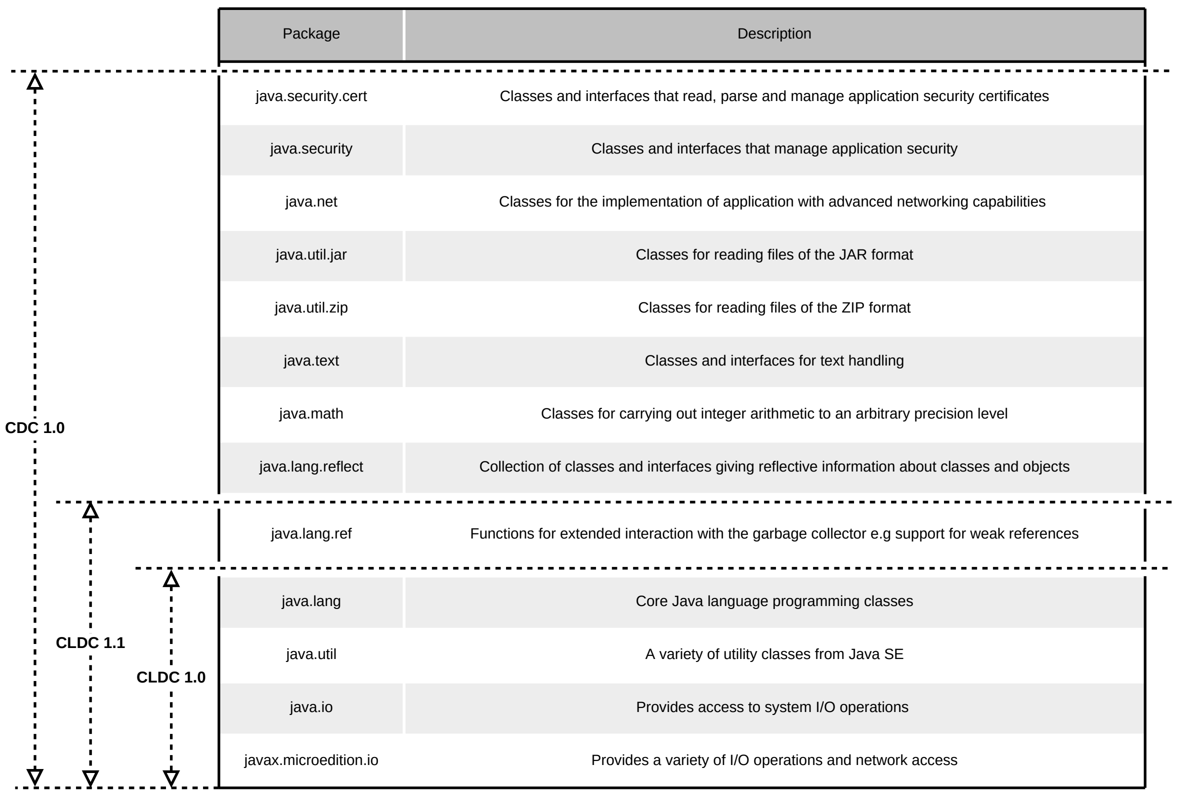
configurations exist: the Connected Device Configuration (CDC) and the Connected, Limited Device Configuration (CLDC). These configurations group mobile devices into two large groups based on device capability as explained below.

3.2.2.1 The Connected Device Configuration (CDC)

This configuration can be implemented by devices that have a minimum of 512KB of read-only memory (ROM), 256KB of random access memory (RAM) and some form of network connectivity [64]. It must also implement a full implementation of the Java Virtual Machine as defined in the Java Virtual Machine Specification, 2nd Edition [64]. Devices that meet these requirements are then referred to as *connected devices*, hence, the *Connected Device* Configuration. Connected devices refer to devices that are not as capable as desktop computers but still have ample memory, processing power and network connectivity. A typical connected device would have a 32-bit CPU as well as 2 MB of RAM and ROM [5]. Examples of connected devices include high-end PDAs and set-top boxes. Current mobile phones do not qualify as connected devices since they are not powerful enough to host the components of the CDC.

Unlike Java SE and Java EE that run on the JVM, the CDC runs on a virtual machine known as the C Virtual Machine (CVM). This VM provides all the same functionality as the JVM but is smaller in size and provides a far superior performance to the JVM [5]. It manages memory efficiently and runs effectively on 32-bit processors, as required for connected devices [91].

The CDC exists in two versions: version 1.0 and 1.1. We will not focus on the CDC any further since this configuration is not suited to mobile phones and is not used in our project. A summary of the packages that the CDC 1.0 implements is illustrated in Figure 3.3.


CDC 1.0
CLDC 1.1
CLDC 1.0

Package	Description
java.security.cert	Classes and interfaces that read, parse and manage application security certificates
java.security	Classes and interfaces that manage application security
java.net	Classes for the implementation of application with advanced networking capabilities
java.util.jar	Classes for reading files of the JAR format
java.util.zip	Classes for reading files of the ZIP format
java.text	Classes and interfaces for text handling
java.math	Classes for carrying out integer arithmetic to an arbitrary precision level
java.lang.reflect	Collection of classes and interfaces giving reflective information about classes and objects
java.lang.ref	Functions for extended interaction with the garbage collector e.g support for weak references
java.lang	Core Java language programming classes
java.util	A variety of utility classes from Java SE
java.io	Provides access to system I/O operations
javax.microedition.io	Provides a variety of I/O operations and network access

FIGURE 3.3: The packages implemented by and the comparison between the CLDC 1.0, CLDC 1.1 and CDC 1.0.

3.2.2.2 The Connected, Limited Device Configuration (CLDC)

This configuration is a subset of the CDC [93] and was designed for devices implementing the Java Micro Edition but that are not powerful enough to implement the CDC. Two versions of the CLDC exist, version 1.0 and 1.1. The CLDC 1.0 can be implemented by devices that meet the following requirements [62] [73]:

- Have between 160KB and 512KB total memory available with 256KB or less ROM/Flash memory and 256KB or less RAM.
- A limited power source (battery or otherwise).
- Some form of connectivity to some type of network but with smaller bandwidth than that of CDC.
- Varying degrees of sophistication as regards their user interfaces, and possibly not having any at all.

The CLDC 1.1 extended the CLDC 1.0. Several features were added and amended [63] [93]. The most important of these changes are the following[63] [93]:

- Support for floating-point data types has been added. The classes, Double and Float, have been added and methods have been added to various other classes such as String to allow, among other things, for conversions to and from the floating-point data type classes. A very important advantage of this addition is that it makes it possible to display 3D graphics on the device.
- The minimum memory budget has been raised from 160KB to 192KB to support floating point arithmetic.
- Support for weak references[1] has been added.

Devices that meet these requirements are then referred to as *connected limited devices*, since they have very limited resources, hence, the *Connected, Limited* Device Configuration. Current mobile phones and pagers are connected limited devices. The CLDC implements a VM known as the K Virtual Machine (KVM) [122]. The 'K' in KVM stands for kilo and is an indicator of the fact that the memory usage of this VM is measured in kilobytes [122] [5] [73]. This VM was made to include the key features of the JVM but with improvements made in the following key areas [122] [5] [73]:

- Reduction and optimization of the size of the VM.
- Severely minimize the memory usage of the VM.
- Modularization of the framework for ease of extensibility and to allow it to be selectively configured for use with different devices.

[1] A reference to a memory location that does not keep it protected from garbage collection. An example of a structure that uses weak references is a list to keep track of the variables currently in use in an application. References to these variables by the list must be weak so that they do not stop the process of garbage collection once other references to these variables have been removed.

The packages implemented by the CLDC 1.0 and CLDC 1.1 as well as their place in relation to the CDC superset are illustrated in Figure 3.3. Our mobile application makes use of the CLDC 1.1 since this configuration, as has been mentioned, provides support for floating point data types necessary for the Mobile 3D Graphics (M3G) API to display our 3D Avatar. The M3G API is explained later in this chapter.

3.2.3 Profiles

A profile is layered on top of and extends a configuration. It adds a set of APIs to a configuration that aid in application development for a specific family of devices. Examples include APIs for user interface (UI) creation, persistent storage and multimedia rendering. Built on top of the CDC are three profiles, namely, the Foundation profile, the Personal Basis profile and the Personal profile, only one of which need be implemented as seen in Figure 3.2. The reason for this will be explained.

The most prevalent, popular and ubiquitous profile built on top of the CLDC is the Mobile Information Device profile (MIDP) [92] [93]. This profile is designed for mobile phones and has been implemented by millions of mobile phones world wide [73]. This is the profile used in our implementation. Other profiles also exist to accommodate other device types such as the Information Module profile (IMP) [60] [43] and the Digital Set Top Box profile (DSTBP) [61]. The following subsections provide an overview of the profiles listed. The first two subsections briefly describe the CDC and CLDC profiles that are not used in our implementation. The section thereafter describes the MIDP in more detail.

3.2.3.1 The CDC Profiles

The Foundation profile [121] complements the CDC as seen in Figure 3.2. It provides additional packages that provide non-Graphical User Interface (GUI)-related functionality. Many of these packages are dedicated to providing increased security to an application such as the encryption of data and setting up secure socket connections.

The Personal Basis profile [123] is a superset of the Foundation profile. It provides all the packages and functionality of the Foundation profile and adds packages for the development of light-weight GUIs as well as the ability to develop and run Xlet[2] applications.

The Personal profile [124] is a superset of the Personal Basis profile. It provides all of the packages and functionality of the Personal Basis profile and provides the full

[2] A Java program designed to run on and manage set-top boxes for Digital television. Xlets differ from Applets in that they can be paused and resumed in runtime.

Abstract Windowing Toolkit (AWT) [120] GUI rendering library. It also supports the development and running of Applets.

Due to the fact that these profiles provide an increased amount of backward-compatible functionality, only one need be implemented, depending on the capabilities of the target device. These profiles are not used in our implementation and are not discussed any further.

3.2.3.2 The CLDC Profiles

The Information Module profile (IMP) is aimed at devices that have similar capabilities to MIDP devices described in the next subsection, but have alternative forms of GUI display or no GUI display at all [60] [43]. Examples include vending machines and tracking devices [43]. It is a direct subset of the MIDP, including all the features of the MIDP with the MIDP graphics rendering library – the LCDUI library – removed. It provides no features additional to the MIDP.

The Digital Set Top Box profile (DSTBP) is aimed at small-footprint television set top boxes, hence, the name. It includes APIs such as those used to provide I/O operations and the location of resources from a Uniform Resource Locator (URL), networking capabilities and graphics rendering capabilities in a manner that is appropriate for television set top boxes. A subset of the Java TV API is also included. The Java TV API, like the CDC-based Personal Basis profile, provides support for an Xlet application programming model appropriate for such devices.

3.2.3.3 The Mobile Information Device Profile

The Mobile Information Device profile (MIDP) is aimed at embedded devices such as mobile phones and PDAs. It caters for devices that are, in one way or another, more capable than IMP- and DSTBP-compliant devices. Three versions of the MIDP currently exist: versions 1.0, 2.0 and 3.0. Figure 3.4 summarizes the packages contained in each of these versions. The MIDP 1.0 was developed under the JSR 37 and is aimed at devices that have [65]:

- A total memory of 512 KB (ROM and RAM) available to the Java runtime and libraries.
- A limited power source. This is usually a battery.
- A limited-bandwidth connection to some form of wireless network.

- User interfaces of some form that can be of varied sophistication.

Devices with these specifications are then referred to as Mobile Information devices [65]. The MIDP 1.0 provided the most basic necessities of a complete application runtime environment. It provides APIs for displaying and managing a GUI suitable for limited depth and size displays (the *lcdui* package), carry out persistent storage and retrieval of information using record management system (RMS) stores (the *rms* package) and provide a basis on which to build and manage a complete application (the *midlet* package). It also provides the ability to interface with user input systems on such devices such as touchscreens and keypads, and setup and use wireless network connections. Applications built on this profile are called MIDlets.

The MIDP 2.0 was developed under the JSR 118 and extended the MIDP 1.0 vastly. It extended the networking capabilities previously provided by adding support for UDP datagrams and TCP socket streams. It also added support for secure (HTTPS), push-initiated and serial connections. Furthermore, it extended the graphics library (the *lcdui* package) and added a new package dedicated to the creation of two-dimensional (2D) games. This new library allows for the creation of gaming canvases and sprites, among other things. The newly added *media* and *media.control* packages provided the ability to render and control sound, including, for example, volume and tone control. A robust security API (*pki*) was also added. This handles security as regards the use of, for example, network connections. For a complete and detailed list of the additions made by the MIDP 2.0, the reader is referred to the article by Knudsen [74].

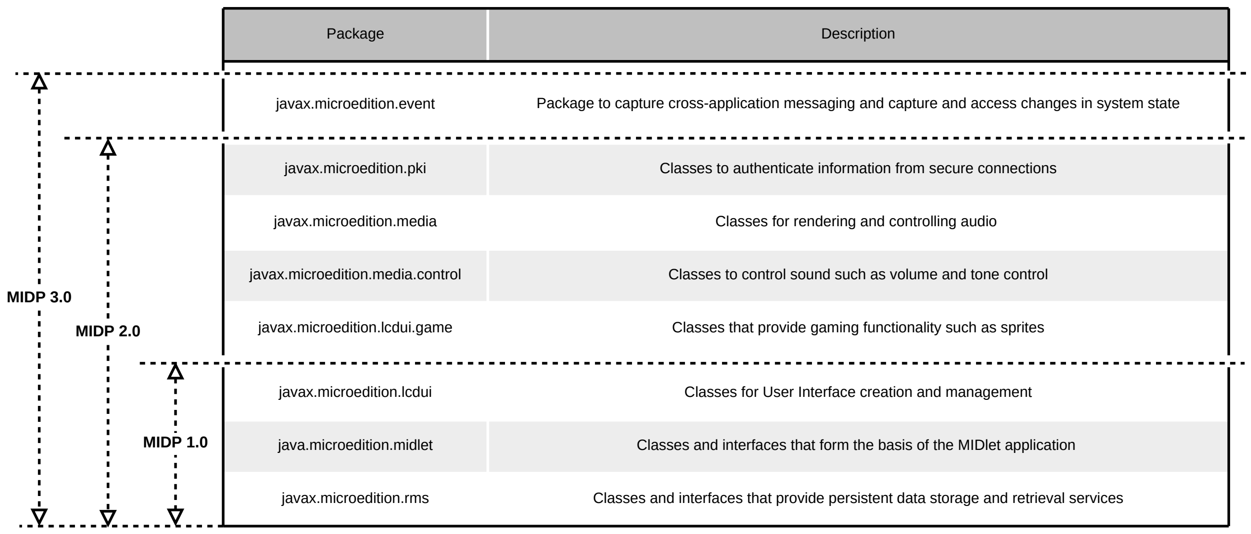
The MIDP 3.0, a very recent development (December of 2009), was developed under the JSR 271 and made further advancements to the MIDP family. A new package has been added (*event*) that provides enhanced MIDlet application control and cross MIDlet communication. The following list summarizes some of the improvements and additions made:

- Enhanced application control:
  - Enable inter-MIDlet communication to take place.
  - Support daemon MIDlets – MIDlets that run in the background and have no UI.
  - Support auto-launch MIDlets – MIDlets that automatically start up when the platform boots up.
  - Support the concurrent running of multiple MIDlets on one VM.
  - Better manage the MIDlet lifecycle and allow firewalling capabilities.

- Provide greater UI rendering capabilities, catering for more sophisticated displays.
- Provide greater functionality for gaming.
- Support secure RMS stores and remote and removable RMS stores.
- Support for the Internet Protocol version 6 (IPv6).

It also amends the MIDP 2.0 and attempts to standardize the specification for improved cross-device interoperability. A complete and detailed list of enhancements made by the MIDP 3.0 is provided by the article by Hopkins [50]. As mentioned, it is a very recent development and has not, as yet, been implemented on any notable or usable scale.

The MIDP 2.0 is currently the most popular CLDC profile, having been implemented by over 2 billion mobile phones worldwide [50]. The MIDP 3.0 is not widely adopted as yet. The MIDP 1.0 does not provide the *media* and *media.control* packages that we required in order to be able to use the Mobile Media API Optional API for the capturing and rendering of video and audio. Therefore, we made use of the MIDP 2.0 in our implementation, since this provides all the functionality we required and is widely supported.


MIDP 3.0
MIDP 2.0
MIDP 1.0

Package	Description
javax.microedition.event	Package to capture cross-application messaging and capture and access changes in system state
javax.microedition.pki	Classes to authenticate information from secure connections
javax.microedition.media	Classes for rendering and controlling audio
javax.microedition.media.control	Classes to control sound such as volume and tone control
javax.microedition.lcdui.game	Classes that provide gaming functionality such as sprites
javax.microedition.lcdui	Classes for User Interface creation and management
java.microedition.midlet	Classes and interfaces that form the basis of the MIDlet application
javax.microedition.rms	Classes and interfaces that provide persistent data storage and retrieval services

Figure 3.4: The packages implemented by and the comparison between the MIDP 1.0, MIDP 2.0 and MIDP 3.0.

3.2.4 Additional APIs

Optional APIs provide additional functionality and access to special hardware capabilities. Examples are the Java APIs for Bluetooth – JSR 82 – which provide access to the device's Bluetooth adapter and the Mobile 3D Graphics (M3G) API – JSR 184 – which enables rendering of 3D graphics by accessing the graphics adapter. These APIs are not central or core to creating MIDlets but provide additional functionality. Even when a device has the necessary hardware capabilities, it remains at the discretion of a device manufacturer to implement the respective additional API, providing Java ME access to that hardware. Table 3.1 on page 29 lists important additional APIs. The two most important additional APIs that are used in our implementation are the Mobile Media API – JSR 135 – and the M3G API – JSR 184. The former is used to record and render video and audio while the latter is used to render our 3D Sign Language Avatar. The following two sections provide some details about these two APIs.

3.2.4.1 The Mobile Media API

The Mobile Media API (MMAPI) [58] builds on the MIDP 2.0 packages *media* and *media.control* (see Figure 3.4) and extends those packages. It also provides an additional package *media.protocol* that allows the creation and use of custom protocols. It provides the following extensions to the MIDP 2.0 media API:

- It added support for video and graphics capture/rendering. The MIDP 2.0 subset was audio-only capable.
- It supports the synchronizing of multiple Players using a common time base. Thus multiple Players may start at an exact time.
- It supports the creation and use of custom protocols.
- The Manager class has been extended and provides, among other things, support for more media types.

The MMAPI makes use of four main classes, namely:

1. Manager
2. Player
3. Control
4. PlayerListener

The Manager class is the first class invoked when accessing media services. It is responsible for creating a working instance of the Player class which will capture/render the content required. Typically, the static method createPlayer of the Manager class is invoked, having passed into it a Uniform Resource Identifier (URI) or InputStream of data, and an optional MIME type for the data requested. The URI determines whether a capture or render operation will take place and the source from which to receive input. The general format of a simplified URI is "[*protocol*] : //[*location*]".

For a capture operation, the protocol is set to "capture". The location is then set to either "audio" for audio capture or "video" for video capture with audio. While this is the standard method for carrying out a capture operation according to the JSR specification [58], not all manufacturers abide by this implementation style. Sony Ericsson mobile phones, for example, differ from the standard and use a location of "audio_video" for video capture with audio. Using a location of "video" on these phones causes a runtime error.

When retrieving a resource, the protocol may be set to a variety of protocols such as "http" for internet resources, "file" for resources on the mobile phone's filesystem and "rtsp" for real-time streaming resources, to name but a few. The location will be set to the location of the file to be retrieved.

The getPlayer method returns an instance of a Player class. The instance of this class provides methods that allow control over the media such as starting, pausing and stopping the media. It also has a setPlayerListener method. This method can be passed an object implementing the PlayerListener interface. That object will then be able to capture and respond to state changes in the Player. The states that a Player goes through are sequential. When the Player is first created it is in the *Unrealized* state in which the Player does not have enough information to gather the required resources to process the media. Resources include the media that is to be captured/rendered and hardware resources such as the speaker, processor and video adapter. Thereafter, a sequence of methods in the Player class are invoked to move the Player forwards through the states until it processes the media and is eventually closed. A complete list of these states, the methods used to take the Player into those states and the description of those states is provided in Table 3.2.

The PlayerListener assigned to the Player can then capture and respond to these states as required. For example, a label may inform the user that the Player is initializing when the Player is in states before the *Started* state and a play button can turn into a pause button when the Player enters the *Started* state.

The Player class also provides a getControl method which takes in a parameter specifying the type of Control requested and returns an instance of that Control. Controls enable the manipulation of the content being captured/rendered. Many different Control types exist that serve various functions. A list of Control types and their descriptions is

Table 3.2: Player states, their descriptions and the methods used to invoke them. The states occur sequentially starting at Unrealized and ending at Closed.

Player State	Invoker Method	Description
Unrealized	-	Player does not have enough information to gather necessary resources to process the media
Realized	realize()	Player has obtained the necessary information to acquire necessary resources
Prefetched	prefetch()	Player has obtained all necessary resources. It is ready to carry out its tasks
Started	start()	Player is processing the content required
Closed	deallocate() close()	Player has released all its resources. It has also stopped processing the content

provided in Table 3.3. The type of Controls available varies with the type of media that is being captured or rendered. Additionally, whether or not all, some or any of the Controls listed have been implemented and are available on a particular device is at the discretion of the manufacturer.

Table 3.3: Standard Controls in the MMAPI

Control	Description
FramePositioningControl	A Control to access individual frames in a video
GUIControl	A Control for display-dependent data such as video
MetaDataControl	A Control to access metadata information contained in a stream such as the author and copyrights
MIDIControl	A Control that provides access to a device's MIDI player
PitchControl	A Control to set the audio pitch
RateControl	A Control to set the playback rate of the media
RecordControl	A Control that allows control over data being captured such as storage location
StopTimeControl	A Control that sets a preset amount of time after which the Player is automatically stopped
TempoControl	A Control to set the tempo of an audio player, usually a MIDI Player
ToneControl	A Control to play monotonic tone sequences
VideoControl	This builds on GUIControl. It controls the display of video
VolumeControl	A Control to set the volume of audio

We use this optional API in our implementation of the capturing of Sign Language video and English Speech. We also use it to render Sign Language video.

3.2.4.2 The Mobile 3D Graphics API

The Mobile 3D Graphics (M3G) API [59] is used in conjunction with the CLDC 1.1 and either MIDP 1.0 or MIDP 2.0. The CLDC 1.1 provides the floating-point data types *Float* and *Double* required for 3D computations. Both MIDPs provide the *lcdui* package which contains a class required by the API – the Graphics class. Its specification was based heavily on the first version of the OpenGL for Embedded Systems (OpenGL ES [71]) graphics library which is a version of the popular OpenGL [72] 3D Graphics library, reduced to run on embedded devices such as mobile phones and PDAs. However, as is the case with all other JSRs, it remains at the discretion of device manufacturers to actually use the OpenGL ES library in their M3G implementations.

The API is contained in a single package, the *m3g* package. This package contains 30 classes of varied function. Table 3.4 lists key classes in the API that were of use in our implementation and has been taken from the official documentation of the API. A complete specification may be found at the official JCP website [59]. It also makes use of the Graphics class in the *lcdui* package provided by the MIDP.

The basic structure of any M3G program is a continuous sequence of four steps:

1. Obtain the Graphics3D singleton instance.
2. Bind a target 2D buffer to the Graphics3D instance.
3. Render objects to the Graphics3D instance.
4. Release the target from the Graphics3D instance, thus, writing the 3D display to the 2D buffer.

As can be seen in the listing, all the steps in this basic structure involve the Graphics3D class. This class is the most fundamental class in the M3G API. It provides the only means to carry out rendering operations. Only one instance of this class exists for each program execution – it is a singleton instance. All functions needed to carry out rendering are accessed through this instance. Table 3.5 lists and briefly describes key methods accessed via the Graphics3D instance. These methods are described below.

Obtaining the singleton instance of the Graphics3D class is done by invoking the static function getInstance() of the same class. The function then returns the singleton instance required. The instance then requires that a 2D buffer in the form of a Graphics object or Image2D object be bound to it. This buffer will be updated with the 2D projection of the 3D graphics rendered, as viewed from the device screen. This process is carried out by invoking the bindTarget() function of the Graphics3D instance and passing a Graphics or Image2D object to it.

Table 3.4: Key classes in the M3G API.

Class	**Description**
AnimationController	Controls the position, speed, and weight of an animation sequence. For example, it can be used to control the blinking and movement of a light in an animation application.
AnimationTrack	Associates a KeyframeSequence with an AnimationController and an animatable property, which is a scalar or vector variable that the animation system can update directly.
Camera	A scene graph node that defines the position of the viewer in the scene and the projection from 3D to 2D. The camera always faces towards the negative end of the Z axis (0, 0, -1).
Graphics3D	A singleton 3D graphics context that can be bound to a rendering target. All rendering is done through the render() methods in this class.
KeyframeSequence	Encapsulates animation data as a sequence of time-stamped, vector-valued keyframes, each of which represents the value of an animated quantity at a specified instant. Can be associated with multiple animation targets.
Light	A scene graph node that represents different kinds of light sources, which are used to determine the color of each object, according to its Material attributes.
Loader	Downloads and deserializes graph nodes and node components, as well as entire scene graphs. Downloading ready-made pieces of 3D content from an M3G file is generally the most convenient way for an application to create and populate a 3D scene.

Continued on next page

Table 3.4 – continued from previous page

Class	**Description**
Mesh	A scene graph node that represents a 3D object defined as a polygonal surface. It represents a conventional rigid body mesh, and its subclasses MorphingMesh and SkinnedMesh extend it with capabilities to transform vertices independently of each other.
Object3D	An abstract base class for all objects that can be part of a 3D world. These include the world itself, other scene graph nodes, animations, textures, and so on. Everything in the API is an Object3D except Graphics3D, Loader, RayIntersection, and Transform.
World	A special Group node that is a top-level container for scene graphs. A scene graph is constructed from a hierarchy of nodes. In a complete scene graph, all nodes are ultimately connected to each other by a common root, which is a World node.

3D graphics are then rendered to the Graphics3D instance. The instance provides four different render methods that correspond to two rendering modes: immediate mode and retained mode. Retained mode is an extension that M3G made over and above the capabilities of OpenGL ES. In retained mode, an entire world is rendered to the Graphics3D instance. This world is created in the form of a tree structure. It may contain cameras, lights and objects such as spheres, cubes, lines, and other such objects. Objects may also be grouped into sets of objects using Group nodes. Each of these objects have attributes such as position, orientation and appearance. The World node is the root node and all other objects are attached to this node or its children nodes. This type of structure is known as a scene graph. A very simplified scene graph is depicted in Figure 3.5.

Any number of additional objects may be added to this world. This is done by creating new instances of those objects using the classes provided in the API such as Camera, Light, Sprite3D and Mesh. Table 3.4 lists key classes in the M3G API. The new objects are then added to the World node or one of its children nodes. This representation of the 3D world provides a high-level abstracted level of control over the objects in that world. To render the world, one of the four render methods mentioned before is used. This render method takes only one parameter – the World node. It then displays everything within that world from the viewpoint of the active camera in that world. The display of this world is then retained from frame to frame and manipulations can be made to it to change the display as required. Retained mode greatly simplifies 3D graphics creation. It has an intuitive nature and encapsulates many low-level display details, providing high-level access to them.

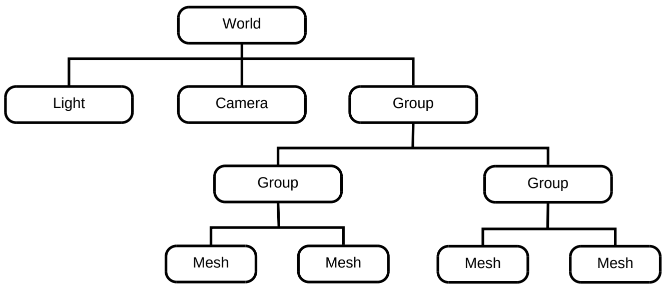


FIGURE 3.5: Graphical depiction of a simplified scene graph.

Immediate mode is a drawing mode that is inherited from the underlying OpenGL ES specification. In this mode, 3D objects are rendered individually and directly to the Graphics3D instance. The remaining three render methods of the Graphics3D instance render individual 3D objects in the form of nodes, or vertex lists. Lights are added directly to the Graphics3D instance using the addLight method. Also, an active camera has to be specified by invoking the setCamera method. All additions and changes made in this mode take immediate effect on the display since they are made to the Graphics3D object directly. The graphics content of each and every frame must be specified in order to maintain a persistent display. This approach provides fine-grained low-level control over the display.

Lastly, the target is to be released. This is done by invoking the releaseTarget method. This causes the 3D display to be written to the 2D buffer and displayed on the device screen.

TABLE 3.5: Key methods provided by the Graphics3D instance.

Method	**Description**
addLight	Adds a light for immediate mode rendering
bindTarget	Sets a target Graphics or Image2D object to render graphics to. These objects serve as a 2D buffer that will be written to the device screen
getInstance	Returns the singleton Graphics3D instance for the currently running program
releaseTarget	Releases the bound target, thus, flushing the rendered 3D display to the 2D buffer
render	Renders the World, Node or sub-mesh
setCamera	Set the active camera for immediate mode rendering

M3G comes with a standard export format, the M3G format. 3D modelling tools can be used to design complex animated 3D scenes. These scenes can then be exported in the M3G format which is a scene graph representation of the scene. All standard 3D modelling tools such as Blender and Maya3D have M3G exporters. The Loader class in the M3G API then provides invaluable functions that allow the M3G file to be imported as a scene graph into M3G. This is a trivial process of making a call to the static function load of the Loader class, having passed into it a path to the M3G-formatted file. The function reconstructs the world and returns the World node of the resulting scene graph. This can then be rendered in retained mode as has already been explained.

This is the exact process that we used in the implementation of our 3D Avatar. This feature of M3G made it highly suitable to our needs. Our Avatar was modelled and animated in Blender, exported in the M3G format, and imported into M3G, and subsequently rendered and animated on the mobile phone.

3.3 Summary and Conclusions

In this chapter we described the Java ME programming language. We explained what it is and where it fits in with other Java versions. We described the Java Community Process (JCP) which develops Java ME APIs. We described the Java ME architecture which comprises of three layers: Configurations, Profiles and Optional APIs. We explained in detail what each of these are. We also gave relevant examples of each, namely, the Connected, Limited Device Configuration (CLDC), the Mobile Device Information profile (MIDP), the Mobile Media API (MMAPI), which is an Optional API, and the Mobile 3D Graphics (M3G) API, which is also an Optional API, all of which were used in our implementation.

At this point we state that the Sony Ericsson C905 Java ME implementation provides the configuration, profile and optional APIs that we have pointed out are necessary for our implementation [113]. We have, thus, selected this phone as our implementation target.

Chapter 4

Sign Language Avatar Creation

As mentioned in Chapter 1, three of the four methods we use to visualize SASL and investigate SASL intelligibility use 3D humanoid Avatars. Two of our methods make use of Van Wyk's Avatar [146] that he called "*Man*". Henceforth, we refer to this Avatar as *Man*. Our third method makes use of a less detailed Avatar that we developed based on *Man* that we call "*Phlank*". Henceforth, we refer to our Avatar as *Phlank*.

In this chapter we explain the methodology, techniques and tools used in the process of Avatar creation. We focus on those techniques and tools that apply to the current research.

Section 4.1 is a generic methodology for the creation of SL Avatars. It details the steps involved, as well as the techniques used to achieve those steps. Section 4.2 then describes the tool Blender which may be used with the SL Avatar creation methodology to create SL Avatars. Van Wyk used Blender to create *Man* and we used it to create *Phlank*. Section 4.3 then provides an overview of the SL Avatar creation methodology that Van Wyk proposed [146] and explain how it was used to implement *Man*. We also explain *Man*'s performance in light of experiments conducted with it.

4.1 SL Avatar Creation

The creation of SL Avatars takes place in steps. The first step involves creating or acquiring a Virtual Human (VH) model. The second step requires the parameterization of the model in order to be able to apply deformation to it for animation. The third step is the creation of SL animation databases for the model for various SL signs. The following subsections provide an overview of the steps, as well as the techniques used to achieve each of these steps. We mainly focus on those techniques relevant to the current research, but also provide overviews of other techniques. Section 4.1.4 then provides an overview of existing technologies and standards that may be used to facilitate specific parts of the process.

4.1.1 Avatar Modelling and Acquisition

Arriving at a VH model that will represent the SL Avatar may involve the creation of the model or parts of the model and/or acquisition of the model or parts of the model from a pre-made model or by means of 3D scanning (see Section 4.1.1.3). In other words, this process falls in between two extremes of: modelling the whole VH from scratch; and obtaining the whole VH from a pre-made source or by means of 3D scanning. In our research, we acquire *Man* from Van Wyk's work, and model part of *Phlank* from scratch and acquire part of it from *Man*, as is explained in Chapter 5.

There are set techniques for modelling Avatars. These are: interactive modelling; parametric modelling; procedural modelling; photogrammetry; and 3D scanning. Our research only makes use of the first two techniques and we focus on these two in the following two subsections. Subsection 4.1.1.3 then provides a brief overview of the other three methods which we do not use.

4.1.1.1 Interactive Modelling

Interactive modelling is a hands-on approach to modelling. It involves the creation and manipulation of primitives such as vertices, lines, polygons, curves and Non-Uniform Rational B-Splines (NURBS) to model 3D objects [131]. There are many techniques used to guide this process. These techniques are mostly combinations of extrusion modelling and box modelling. In both techniques, the artist starts with a simple 3D object such as a cube.

In extrusion modelling, the initial object is extruded outwards in one or more directions repeatedly, modelling the overall shape of the target object. In box modelling, the initial object is subdivided and cut to mould the target object. These techniques may be used inclusively to model various parts of the Avatar.

When using this technique, it may be challenging to produce models that are smooth and have a large number of vertices, that is, are highly refined. Methods and algorithms have been devised to automate a process of increasing the number of vertices on a model, making it smoother and more refined. This process is known as subdivision surfacing. Three well-known methods that carry out this task are the Catmull-Clark method [22], the Doo-Sabin method [31] and Loop's method [77]. Catmull-Clark subdivision surfacing allows for variable levels of subdivision starting at level 1. The higher the level, the more refined the model.

The greatest advantage of interactive modelling is that it provides absolute control over the modelling process and the results obtained. On the other hand, it is an art and requires artistic skills to realize good results. It is also much more time consuming than other methods.

4.1.1.2 Parametric Modelling

Parametric modelling involves the manipulation of an existing parameterized model, known as a template model, to form the desired model. The template model may have been created or acquired using other techniques. It is divided into segments and parameterized with adjustable attributes such as length, width, height and weight. These attributes have maximum and minimum values. Varying attribute values then affect the appearance of the model.

Two methods have been used to tie attribute values to appearance changes in the model. The first method is the use of a database of scanned models of varied attributes. These are used to construct a target vector space bound to the provided attributes. Manipulation of the attributes transforms the template model into a target in the vector space [107]. The second method makes use of morphing. The maximum and minimum attributes are tied to a database of modelled maximum and minimum morph targets. Manipulation of the attributes then transform parts of the template model to the modelled morph targets [13].

Parametric modelling is a quick, easy and intuitive process. In this respect, it is very user-friendly. It requires minimal skills on the part of the modeller. It can be carried out using a parametric modelling tool, the most prominent of which are the open-source tool MakeHuman [79] and the proprietary tool Poser [112]. On the other hand, it provides a limited degree of modelling freedom. The modeller is completely bound by the attributes, attribute ranges and the target models provided. It also takes very long periods of time, typically years, to develop a template model and target database.

4.1.1.3 Other Modelling Techniques

The following are three additional modelling techniques not relevant to our research:

- Procedural modelling: This technique generates models by generating graphics from a set of base rules describing the model. The modeller describes the target model or parts of the target model in terms of a set of instructions that may be executed sequentially to generate the target model or its parts. Most modern interactive modelling software provide interfaces to programming languages such as Python or VBScript. These programming languages can then be used for procedural modelling. The advantage of this technique is that it can be used to produce well-defined, re-usable, flexible, parameterized scripts of small size for model creation. On the other hand, it can be a complicated technique to use since it provides no instant visual feedback mechanism. Watt and Watt [142] provide more information on this technique.

- Photogrammetry: This technique uses images of a 3D object captured from multiple calibrated well-positioned cameras. The images are then analyzed to determine common points. A line of sight may then be extended from the location of the camera to the point on the object. This is done for all images and the intersection of all such lines of sight determines a three-dimensional point location for that point. This process may be repeated until the location of the three-dimensional points for the entire object have been determined. This process will ideally be automated although it may be done manually. This method is suitable for modelling people. The advantage of this technique is that it is much less expensive than 3D scanning and is relatively easy to carry out when automated. On the other hand, it is unsuitable for modelling features that are not completely visible such as eyes, ears and the tongue. Slama *et al.* [111] and Atkinson [8] provide more information on this technique.

- 3D Scanning: This technique makes use of a 3D laser scanner to scan an object and collect numerous data points that represent the 3D model. While different technologies exist with different hardware, most project a laser onto the object and re-collect the reflected photons to determine the shape of the object. Other characteristics may also be determined such as the colour. The advantage of this technique is that it is a very fast automated modelling method and requires very little work from the operator of the scanning device. It is also the most accurate method of modelling and can obtain results that no other method can. On the other hand, it is by far the most expensive method of modelling. 3D scanners are very costly. It is also not capable of scanning in features such as eyes, ears and the tongue. Also, post-processing of the data obtained may be difficult due to the large amount of data involved. Bernardini and Rushmeier [15] and Böhler and Marbs [18] provide more information on this technique.

4.1.2 Avatar Parameterization and Deformation

After the VH model has been created or acquired, it needs to be animated. Animation of the model involves deforming the model or parts of it by applying geometric transformations such as rotation, scaling or translation to it. In order to achieve this, the model needs to be parameterized. The parameters can then be varied to deform the model in an appropriate fashion using a deformation technique.

Techniques that can be used to parameterize and deform a model are: direct parameterization; morphing; skeletal subspace deformation; and free form deformation. Our research only makes use of skeletal subspace deformation and Subsection 4.1.2.1 focuses on this parameterization and deformation technique. Subsection 4.1.2.2 then provides a brief overview of the other three methods which we do not use.

4.1.2.1 Skeletal Subspace Deformation

This deformation technique is the most popular technique for the parameterization and deformation of models [52]. It is specifically suited to the task of Sign Language visualization [29] [151] [153]. It involves the construction of a skeleton-like structure under the 3D model. The skeleton is primarily made up of a root bone. The skeleton's co-ordinate frame known as the skeleton space is placed at the root bone. A tree of additional bones is then attached to the root bone. Each bone has a local co-ordinate frame known as the bone space and is influenced by the bone spaces of all bones higher up in the tree. Each bone may be transformed by scaling as well as rotation and translation with degrees of freedom (DOFs) that may be adjusted.

Having constructed the skeleton within the model, the model is parameterized by attaching the bones in the skeleton to points on the model. This process is commonly known as “rigging” or “skinning”. It involves defining the degree to which transformations of a bone should influence points on the model by assigning deformation weights to each point. Initial methods only assigned a single weight to each point producing irregular deformations. Later, methods of assigning multiple weights to each point and linearly blending the weights over the model surface were devised. These produced much more natural and smooth deformations. The rigging process can either be done manually by means of a process known as vertex weight painting or automatically by means of an automatic rigging algorithm.

Vertex weight painting involves manually assigning deformation weights to points on the model. Automatic rigging algorithms determine and assign weights automatically. An automatic rigging algorithm relevant to this research is the algorithm by Baran and

Popović. This algorithm determines and assigns deformation weights to points based on the heat equilibrium over the surface of the model. It is fast and produces deformations of very high quality that can be used to produce high quality animations. Baran and Popović [12] provide further information regarding this algorithm. This algorithm has been implemented in and integrated into Blender.

There are many advantages to skeletal subspace deformation. It is very efficient [11] [16], it is very intuitive in Avatar animation since it is a representation of the normal body movements of a human being and it is easy to implement using automatic rigging algorithms provided by many interactive modelling packages. On the other hand, it has shortcomings and is unable to model a real skeleton exactly. Certain undesirable deformations may occur such as the so-called "candy wrapper" and "collapsing elbow" effects. Both of these effects take place when joints are rotated to extreme angles causing the model to experience a collapse in volume which is not characteristic of a human body [52]. The candy wrapper effect takes place when a bone is rotated on the axis of the bone while the collapsing elbow takes place when the bone is rotated perpendicular to the bone axis.

The advantages provided by this method as well as the integrated support for this method by Blender in the form of the algorithm by Baran and Popović weighs heavily in its favour above other parameterization techniques for the parameterization of *Man* and *Phlank*. No such integrated support exists for other parameterization techniques. It is used to rig both *Man* and *Phlank*.

4.1.2.2 Other Parameterization and Deformation Techniques

The following are three other parameterization and deformation techniques not relevant to our research:

- Direct Parameterization: This technique involves selecting a number of key points on the surface of the model that will have transformations applied to them to produce the required deformations. These points may be defined explicitly or by means of a density function that maps onto a region of points. The advantage of this technique is that it is well-suited to the parameterization of facial models since the face has key locations such as the cheeks and eyebrows, that can be manipulated to produce facial expressions and is not very complex. On the other hand, it is not suitable for body animations since it takes long periods of time to parameterize the body correctly. For more information, we refer the reader to the research papers by Parke [94] and Pasquariello and Pelachaud [95].

- Free Form Deformation: This technique places the model inside a volume, usually a parallelepiped. A grid of control points is defined on the volume surface. The control points are then assigned to points on the model surface be means of a triple tensor product Bernstein polynomial [104]. Transformations may be applied to the control points which then affect points on the model. The advantage of this technique is that it provides a good level of control over deformations in the model, depending on the number of control points used. On the other hand, it may be challenging to express animations in this form. The reader is referred to the research paper by Sederberg and Parry [104] for more information on this technique.

- Morphing: In this technique, a "source morph" and "target morph" are defined and stored, and refer to the initial and final positions of the points on the model to realize a required deformation change. Morphing is a definition of the process of applying transforms to the source morph to arrive at the target morph by means of interpolation. This method is capable of producing very high quality facial and body animations. It is very flexible and can be used to define very complex deformations since one is free to define the target morph as one pleases. On the other hand, it can be extremely challenging to define a sufficient number of target morphs, requiring time and energy. For more information on this technique the reader is referred to the research paper by Lee and Magnenat-Thalmann [75].

4.1.3 Avatar Animation

After the VH model has been parameterized, it can be deformed in different ways to create SL animations (and animations in general). Depending on the parameterization and deformation technique used, a database of parameters needs to be constructed that may be used to realize different animations. This database is referred to as an animation database and the data stored in it is called animation data.

Two techniques exist that may be used to construct the animation database, namely, keyframing and motion capturing. The first technique involves creating animation data while the second technique acquires this data. The following subsections describe these two techniques.

4.1.3.1 Keyframing

This technique involves the manual creation of animation data. The animation data may take different forms for different parameterization and deformation techniques and the process of keyframing may differ in each case. For the case of morphing, the animation database consists of target morph models. It is constructed by manually deforming points on the model to various target morphs. Each target morph is then stored in the animation database. For the case of direct parameterization and free form deformation, the animation data consists of control point locations. It is constructed by identifying key poses in the animation, known as keyframes, and manually deforming the model to match those poses. Keyframes are then stored in the animation database along with the time values at which they occur. Interpolation is used to animate between these poses. We do not discuss the animation of these parameterization techniques any further.

We now consider the process of animating a model that has a skeleton and has been parameterized using skeletal subspace deformation. Key poses of the model are identified for a particular SL gesture animation. These are called keyframes. Each bone of the skeleton is manually transformed to a desired location, rotation and size to match the pose in each of these keyframes. The location, rotation and size of each bone is stored corresponding to the time values of each of the keyframes. Spline-based interpolation is used to animate between these poses to create the animation.

Two methods exist that are used to pose the model, namely, forward kinematics (FK) and inverse kinematics (IK). Forward kinematics involves transforming each bone into the correct pose starting at bones higher up in the skeleton tree and working down the tree in the same manner. Inverse kinematics involves transforming a bone lower in the tree and automatically calculating and assigning rotation angles to its parent and ancestor bones such that the bones remain connected and the transformation is valid [87].

Keyframing is advantageous in that it is inexpensive as compared to motion capture and it creates much less animation data than motion capture. On the other hand, it is a far more complex, time consuming and laborious process as compared to motion capture. It also requires additional skills since it involves the use of animation modelling packages.

4.1.3.2 Motion Capture

This technique makes use of highly advanced motion capture equipment to capture poses of a live human performance. The data capture may take place at variable sampling rates, with higher sampling rates producing more refined animation databases. Initial motion capture equipment took the form of full body suits or gloves fitted with potentiometers that could detect the motion of the body or hands. Modern equipment, however, makes use of optical data capture means. Optical markers are placed on key positions on the performer and specialized cameras capture the motion of these markers [118]. The captured data can then be used to construct highly detailed animation databases.

Motion capture produces extremely realistic and accurate animations. However, it requires equipment that is very costly and it is non-trivial to set up and calibrate the equipment correctly.

4.1.4 Existing Standards and Tools

Many standards have been developed to guide the Avatar modelling process. We explain two of these standards, namely, H-Anim and the MPEG-4 Facial Definition Parameters (MPEG-4 FDP). Many tools have also been developed that greatly simplify parts of the process. We focus on the tool MakeHuman. The following subsections provide an overview of these technologies.

4.1.4.1 MakeHuman

MakeHuman [79] is an open-source parametric modelling tool that enables the creation of state-of-the-art [14] 3D humanoid Avatars in a very short period of time, with times as short as two minutes claimed [79]. One of the goals of the project is to ensure that the model is optimized so that it is anatomically correct and accurate with the least number of vertices. It therefore ensures computational efficiency [79]. It provides numerous parameters that can be manipulated to produce many variations of the original model. Examples of the attributes provided are gender, age, muscle tone, weight and stature. It also provides tools to export the model to many standard formats, such as the Wavefront object file format.

It was originally developed by Manuel Bastioni as a Python script embedded in Blender. Due to its complexity, however, it was later developed as a standalone C++ application. The model used in the project has been through several iterations leading up to the current model of MakeHuman (version 1.0) called *HM06*, a state-of-the-art model [79].

MakeHuman version 0.9.0 was used in Van Wyk's implementation of *Man* as is explained in Section 4.3.2.

4.1.4.2 H-Anim

H-Anim, short for Humanoid Animation [3], is an open standard developed by the H-Anim working group [48] to model VHs for the Virtual Reality Markup Language

(VRML) [4] and X3D [2] formats. It specifies a standard implementation for the skeleton structure of humanoid Avatars parameterized using skeletal subspace deformation described in Section 4.1.2.1. Specifically, it fully defines the structure of the tree of bones, defining individual bones and their names, as well as specifying where in the tree of bones they are to be placed. Table 4.1 provides a small sample of bones and the names assigned to them by the H-Anim specification. Figure 4.1, adapted from the H-Anim specification, depicts part of the tree structure and the bone hierarchy. The complete tree is available in the specification [3].

TABLE 4.1: Sample list of H-Anim-assigned names for bones

Bone	**Name**
Sacrum	HumanoidRoot
Skull	skullbase
Right Thigh	r_hip
Left Calf	l_knee
Left Clavicle	l_sternoclavicular
Right Forearm	r_elbow
Left Pinky Metacarpal	l_pinky0

H-Anim was developed to address the need to share animation data between models as well as sharing models themselves between modellers in different parts of the world. H-Anim made it possible to use the same modelling software to view and/or edit H-Anim-compliant models from different modellers [3].

The H-Anim specification defines a term "*Level of Articulation*" (LoA) that ranges from 0 to 3 and refers to standard quantities of joints (articulations) present in a skeleton structure. A LoA 0 skeleton is the lowest LoA and contains only one bone, the root bone known as the HumanoidRoot. LoAs from 1 onwards have arms and hands and an increasing number of bones in the spine, with LoA 3 containing 72 bones in total and imitating a near realistic spine.

The H-Anim specification falls short when it comes to the face and only specifies bones for the jaw, eyeballs, eyebrows, eyelids and one bone for the skull. The specification provides the bare-minimum of what is required in the skeleton and suggests extension where necessary, for example, by suggesting the implementation of the MPEG-4 Facial Definition Parameters on an H-Anim skeleton for facial expressions [3].

4.1.4.3 MPEG-4 Facial Definition Parameters

The MPEG-4 Facial Definition Parameters (MPEG-4 FDP) is a subset of the MPEG-4 Face Body Animation Parameters (MPEG-4 FBAP) [1] which is an ISO standard

```
HumanoidRoot : sacrum
 sacroiliac : pelvis
 |  l_hip : l_thigh
 |    l_knee : l_calf
 |      l_ankle : l_hindfoot
 |        l_subtalar : l_midproximal
 |          l_midtarsal : l_middistal
 |            l_metatarsal : l_forefoot
 |  r_hip : r_thigh
 |    r_knee : r_calf
 |      r_ankle : r_hindfoot
 |        r_subtalar :  r_midproximal
 |          r_midtarsal : r_middistal
 |            r_metatarsal : r_forefoot
 vl5  : l5
   vl4 : l4
     vl3  : l3
       vl2 : l2
         vl1  : l1
           vt12 : t12
             vt11 : t11
               vt10  : t10
                 vt9 : t9
                   vt8 : t8
                     .
                     .
                     .
```

FIGURE 4.1: A segment of the H-Anim specification for the tree of bones, adapted from [3].

that specifies a standard parameterization technique for both the face and body of 3D Avatars. We focus on the FDP subset of the standard.

The MPEG-4 FDP is a direct parameterization specification for the face. It specifies two sets of feature points on the face. The first set are parameter control points (Facial Animation Parameters), the motion of which, represent fundamental facial actions. The second set represents standard face locations. The first set and second set are depicted in Figure 4.2 as red and blue dots, respectively.

The MPEG-4 FDP can be used to produce complex facial expressions including complex mouth shapes.

4.2 Blender – an Open Source 3D Graphics Tool

Blender [16] is a free and open-source generic 3D modelling, animation and rendering tool. It provides numerous features that facilitate the SL Avatar creation process explained in the previous section. We adopt the use of this tool for developing *Phlank*. The following subsections provide an overview of Blender and explain the tools and features it provides that facilitate the SL Avatar creation process.

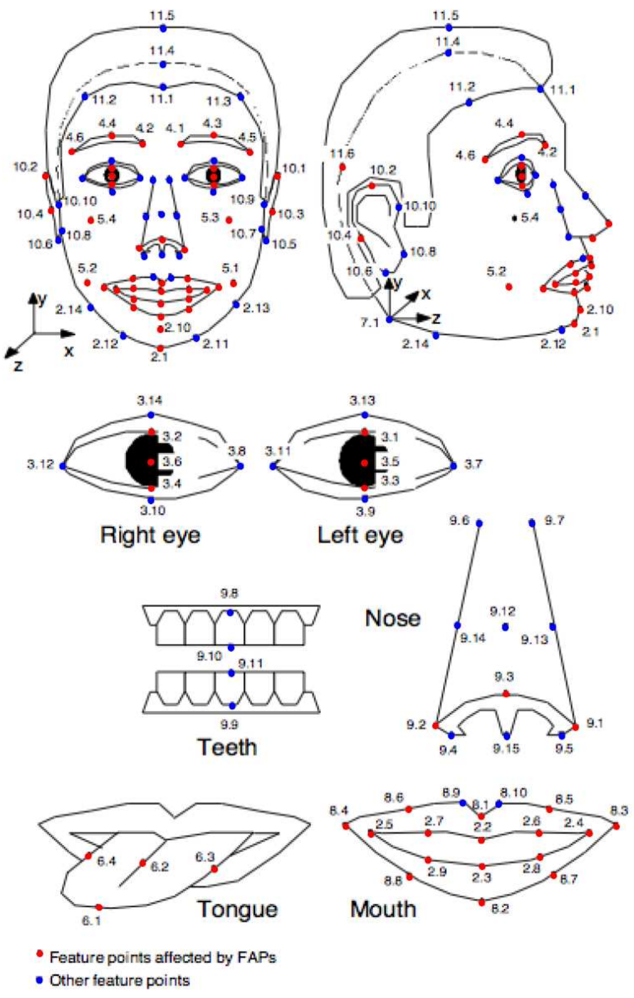


FIGURE 4.2: MPEG-4 Facial Definition Parameter feature points [146].

4.2.1 Background

Blender was developed by the Blender Foundation [16]. It was originally developed closed-source and in-house by the Dutch-based animation studio NeoGeo and Not a Number Technologies (NaN). After NaN went bankrupt in 2002, Blender was released to the public under the terms of the GNU General Public License, making it free and open-source. Since then, Blender has received wide acclaim and is actively being developed on an open-source basis under the supervision of the Blender Foundation [16]. It has a very large user community. This is attributed to, both, the fact that it is free and open-source and that its features are characteristic of high-end 3D graphics modelling software [24] such as Maya and 3ds Max, both commercial software packages.

Blender version 2.48 was used to develop both *Man* and *Phlank*. In the next section we mention some of the tools that this version provides. Towards the completion of this research, a new version of Blender was released, version 2.49. The new version boasts new features and enhancements. For a full specification of these new features, we refer the reader to the official Blender 2.49 website [17].

4.2.2 Tools and Features

Blender provides a vast array of tools that satisfy all our requirements for modelling, animating and exporting SL Avatars. These include:

- **A user-friendly multi-window user interface**. The Blender user interface is fully customizable and may be subdivided, both, horizontally and vertically. Each window may be customized to display any of the available windows in Blender.
- **Advanced character modelling and sculpting tools**. Blender provides a vast array of tools to enable and facilitate the creation of high quality models. It supports the creation of numerous primitives such as vertices, edges, polygons, metaballs, NURBS and curves. It also has sculpting tools that facilitate both box modelling and extrusion modelling such as face, edge and vertex extrusion, 3D path extrusion, and face and edge subdivision. It has an advanced toolset for applying transformations to model primitives such as pivot-based transforms on various axes and transforms with a variety of proportional edit falloffs such as smooth, sphere, root and sharp falloff patterns. Furthermore, it provides tools that automate complicated duplication processes such as spin- and screw-based duplication. In addition, its advanced modifier toolset automates complicated processes such as model subdivision surfacing to obtain more defined models by means of the Catmull-Clark method, model smoothing and mesh mirroring.
- **An advanced armature (skeleton) system**. Blender facilitates skeletal subspace parameterization of models. It provides a skeleton construction toolkit in which an armature represents a bone. Armatures can be scaled, rotated and translated at will. Each armature has a root and tip. The root acts as a joint on which the bone rotates, and the tip serves as a point of articulation for other armatures. Blender also provides tools to apply constraints to the DOF of armatures where necessary and may be applied to, for example preventing a neck bone from rotating 360°. Additional armatures can be added to any other armature to construct a tree of bones. Blender also provides multiple ways to rig a model. It facilitates manual vertex weight painting but also provides automatic rigging algorithms such as the algorithm by Baran and Popović. Rigging a model using this algorithm in Blender involves only a few mouse clicks.

- **Advanced animation tools**. Blender provides advanced tools for keyframing animation. It provides an intuitive interface for posing rigged models. Poses can then be added to a timeline as keyframes with a few simple mouse clicks. Blender then interpolates between keyframes automatically. The timeline can be edited by shifting, duplicating and removing keyframes. Bones on the skeleton can also be grouped to create an orderly workspace, which is especially useful for models with many bones. Blender also provides an IK-solver constraint modifier which may be added to armatures in the skeleton which automatically enables Inverse Kinematics on that armature.

- **Rendering and export capabilities**. Blender provides a built-in rendering engine, the Blender rendering engine, as well as many other rendering engines such as YafRay, Mental Ray, Sunflow and Kerkythea. These provide a variety of rendering capabilities. It also provides exporting capabilities by means of Python scripting to enable the export of models into many standard formats, including M3G, VRML, X3D, WaveFront, and many others.

Blender is therefore very well-suited to our needs.

4.3 Van Wyk's Methodology

Van Wyk's work [146] produced a methodology for the creation of high-quality 3D humanoid Avatars using open technologies and standards. He demonstrated the effectiveness of his methodology by using it to implement his Avatar *Man*. In this section we describe his methodology and its use in the implementation of *Man*. We also summarize the performance of *Man*.

4.3.1 Methodology Overview

Van Wyk's methodology made use of open technologies and standards to facilitate and guide the SL Avatar creation process of Section 4.1. His work proposed the use of Make-Human to create a high quality VH. As explained in Section 4.1.4.1, MakeHuman makes it possible to create a high quality tailor-made Avatar in a very short period of time. He then proposed that the model be rigged using the H-Anim standard for the body and the MPEG-FDP for the face. In this way, the Avatar would be parameterized so as to allow realistic animations of the body and face. He further proposed that an animation database be constructed by means of keyframing. The Avatar can then be animated by calling up animations manually or procedurally. Figure 4.3 is a graphical depiction of this methodology taken from [146]. For a full specification of his methodology, the reader is referred to [146].

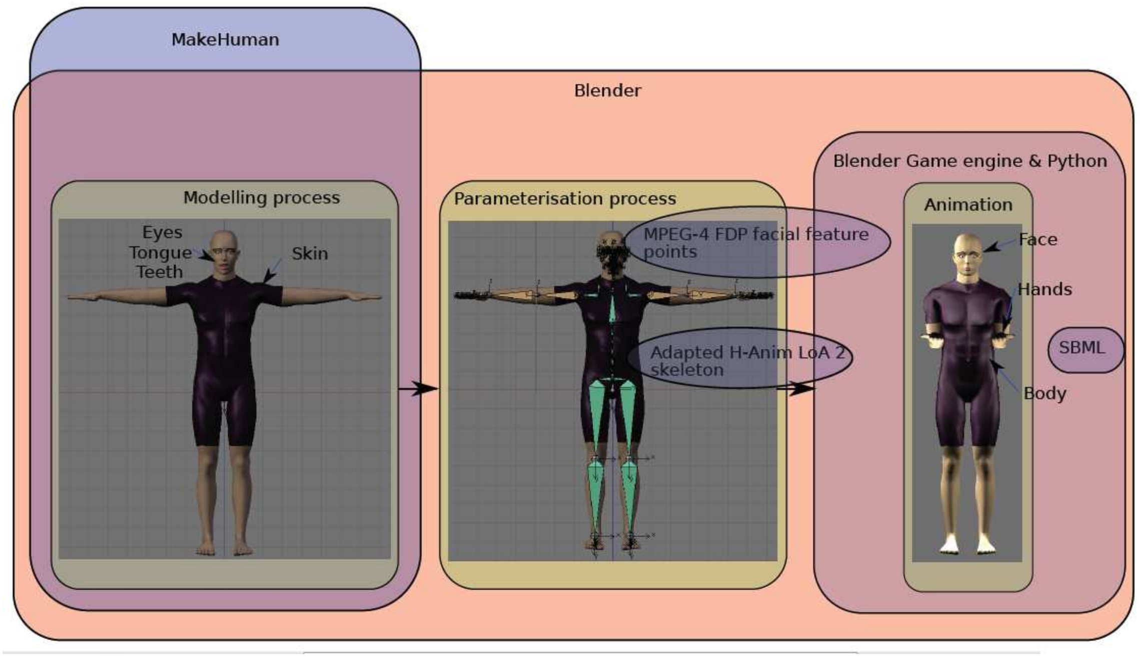


Figure 4.3: Overview of Van Wyk's Methodology, taken from [146].

His methodology is highly advantageous in that it makes use of available technologies and produces a high quality Avatar. It can be used to display high quality SL gestures on desktop PCs. On the other hand, the resultant Avatar is very refined and has many vertices. It requires large amounts of memory and processing power only characteristic of desktop PCs. This makes it difficult, if not impossible, to export it and display it as a 3D model on mobile devices. We explain our attempt at exporting his Avatar and importing it onto a mobile phone in Chapter 5.

We propose that his SASL Avatar system as well as all other processing power- and memory-intensive systems be implemented as services on a client/server architecture. It can be queried to produce SASL animation videos that a mobile device may request and render.

4.3.2 Implementation of *Man*

Using the methodology described in the previous section, Van Wyk implemented his SASL Avatar *Man*.

In modelling his Avatar, he used MakeHuman version 0.9.0. This version of MakeHuman makes use of a model called the K-Mesh. It has 10936 vertices forming 10857 faces, 10387 of which are square faces, and 470, triangular faces. Table 4.2 summarizes the

MakeHuman parameter adjustments he made to the initial model to produce *Man*. By default, the model assumes the "Crucifixion" pose, with hands stretched out to its sides, in accordance with the H-Anim standard [3] and has its jaw closed. The model was left in the *Crucifixion* pose. The jaw was opened because this was required in order to parameterize the face correctly [146]. It was then exported as a WaveFront object file.

TABLE 4.2: MakeHuman parameter values for the creation of *Man*, adapted from [146]

Parameter	**Value**
head baby	0.20
jaw open	0.20
neck muscular	0.30
dorsi muscular	0.50
pectoral muscular	0.20
pectoral forward	0.50
trapezious muscular	0.30
r_shoulder move sideways out	0.40
r_shoulder move sideways in	0.50
l_shoulder move sideways out	0.40
l_shoulder move sideways in	0.50
r_upper arm fat	0.50
l_upper arm fat	0.50
r_lower arm fat	0.30
l_lower arm fat	0.30
abdomen muscular	1.00

The model was then imported into Blender. It was scaled up 30 times. Small importation errors caused incorrect assignment of vertices and he interactively and manually corrected them. The eyeball models were found to be too complicated and were interactively and manually replaced with half-spheres modelled to look like realistic eyes. A different colour material was applied to the torso and upper legs of the model in place of clothes. The eyebrows were similarly created by applying a black material to the correct region on the face of the model.

The model was parameterized using H-Anim. An H-Anim LoA 2 skeleton was fitted inside the model using the H-Anim specification, that is, starting at the HumanoidRoot and adding additional armatures. Armatures were named according to the H-Anim specification as well. Rotational constraints were placed on various bones. Table 4.3 summarizes rotational limits placed on key bones but the reader is referred to Van Wyk's thesis [146] for a complete specification. The completed model is displayed in

Figure 4.4. Figure 4.5 is a close-up of the face of the model showing the MPEG-4 FDP feature points.

TABLE 4.3: Rotational limits placed on specific bones, adapted from [146]

Bone	X-axis range (°)	Y-axis range (°)	Z-axis range (°)
acromio-clavicular	-5 to 50	-13 to 30	-13 to 24
shoulder	-90 to 80	-53 to 158	-45 to 135
elbow	0	-71 to 84	0 to 146
wrist	-73 to 71	0	-33 to 19
thumb3	-90 to 60	0	0
ring0	0	0	-5 to 5
pinky0	0	0	-5 to 5

The face of the model was parameterized using the MPEG-4 FDP specification. Armatures were placed at MPEG-4 FDP feature points. Also, a separate skeleton consisting of 4 armatures was placed inside the tongue model.

The tongue, eyes and teeth were manually attached to their respective armatures by means of vertex weight painting. The algorithm by Baran and Popović was used to attach the skeleton to the remaining parts of the model automatically.

At this point, his Avatar was ready to be animated. He then used keyframing techniques to create animations, although this was not the focus of his work.

4.3.3 Performance

The performance of *Man* in displaying real-time animation was tested [146] on a MacBook Pro with a 2.16 GHz Intel Core 2 Duo processor, 1GB RAM and ATI Mobility Radeon X 1600 graphics card. The geometry details of *Man* are summarized in Table 4.4.

TABLE 4.4: Geometry details of *Man*, adapted from [146]

Section	Faces	Edges	Vertices
Skin, Eyelashes and Lips	9302	18585	9247
Teeth	1140	2291	1215
Tongue	143	294	152

Man was shown to be capable of all necessary poses. The posing results are summarized below:

1. **The spine** was shown to be capable of lateral and forward bending as well as rotation and extension.

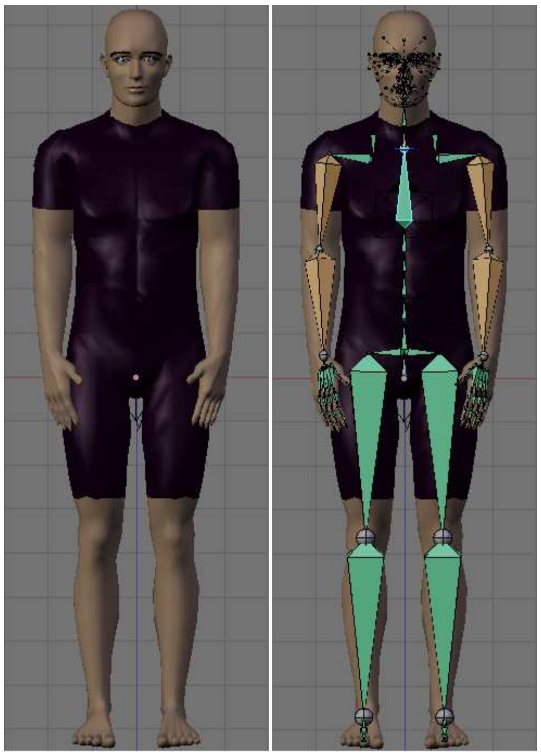


FIGURE 4.4: Completed *Man* model without (left) and with (right) a completed skeleton.

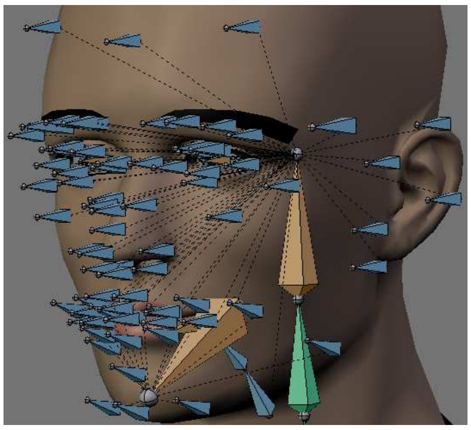


FIGURE 4.5: Completed *Man* model's face depicting MPEG-4 FDP facial feature points.

2. **The neck** was shown to be capable of extension, flexion, rotation and lateral bending.

3. **The shoulder** was shown to be capable of extension, flexion, external and internal rotation in the neutral pose, elevation, abduction, as well as internal and external rotation in abduction.

4. **The elbow** was shown to be capable of supination, pronation and flexion.

5. **The wrist** was shown to be capable of dorsiflexion, palmar flexion, radial deviation and ulnar deviation.

Man was also found to be free from deformation defects such as “Collapsing elbow” and “Candy wrapper” effects, both described in Section 4.1.2.1. *Man* was also animated and the animation frame rate, recorded. The aim was to be able to provide a frame rate for real-time animation which is between 15 and 25 frames per second (fps). It was shown that it could be rendered at between 120 and 230 fps, much higher than the requirement.

4.4 Summary

In this chapter we described a methodology, tools and techniques for the creation of SL Avatars. We explained the steps involved in a generic methodology for SL Avatar creation as well as describing the techniques that may be used to achieve each of the steps. We also described the tool Blender which we showed to be well-suited to the task of SL Avatar creation. We also explained the methodology used to create Van Wyk’s Avatar *Man* and its implementation based on that methodology, as well as its performance.

Chapter 5

Methodology and Implementation of the Avatars

In this chapter we describe the creation methodology of our reduced Avatar *Phlank* as well as the animation and exportation of both *Phlank* and *Man*. *Phlank* was developed specifically to run as a 3D model on the mobile phone. Rendering an Avatar as a 3D model, as opposed to rendering it as video, makes it possible to manipulate its structure and animations in real-time and provides a level of flexibility that is difficult to provide using video.

It was initially hoped that it would be possible to export and render the Avatar *Man* as a 3D model on the mobile phone and we attempted to carry this out. This process is explained in this chapter. It was found that it was not possible to export *Man* and render it on the mobile phone as a 3D model. As a result, we decided to implement a reduced Avatar that could run as a 3D model on the mobile phone and compare the feasibility of the two Avatars. While this was the chronological order of events, this chapter explains the implementation of the two Avatars according to the structure of the generic SL Avatar creation methodology, explained in Chapter 4.

As mentioned in Chapter 2, the research conducted by the MobileASL project [23] showed that the quality of frames in a SL video had a greater impact on SL intelligibility than the frame rate. They found that reductions in frame rate while increasing frame quality enhanced the intelligibility of the SL in the video. This was a major consideration in the implementation of *Phlank*.

The following sections describe the modelling and parameterization of *Phlank* and the animation and exportation of both *Phlank* and *Man*.

5.1 Creation of the Avatar *Phlank*

5.1.1 Modelling and Acquisition

Phlank was specifically designed to be able to run as a 3D model on mobile devices. A model with an excessive number of vertices is not able to run on the mobile phone. An "out of memory" error is reported in such instances. Therefore, a basic Mobile 3D Graphics Java ME application was developed with the sole purpose of periodically evaluating whether or not the model could run at a particular level of detail. This application, called and referred to henceforth as "TestModel", may be found in Appendix A. The model was exported according to the procedure detailed in Section 5.2.2 at key points in the modelling process and imported into TestModel.

In order to provide some consistency between the two Avatars *Man* and *Phlank*, both belonging to the SASL project, we decided to acquire the face of *Man* and model the rest of the *Phlank* interactively. It was also decided that *Phlank* would not be given a body or upper arms since these would add unnecessary detail to the model.

We started by interactively modelling the right arm, assigning dimensions to each part intuitively based on the appearance of the result. A box was added to the scene. It was scaled to dimensions that were suitable for the forearm, a length, width and height of 0.10, 0.42 and 0.810 Blender Units (BUs) respectively. We then used an extrusion modelling technique to map out the general structure of the remaining parts of the arm and hand. The arm was repeatedly extruded outwards to form one segment for the forearm, three segments for the glove strap, one pre-palm segment and one segment for the palm. The number of BUs by which each segment was extruded is summarized in Table 5.1. The resultant model can be seen in Figure 5.1. Note that two of the glove strap segments and the pre-palm segment are not clearly visible due to their small height. Figure 5.2 provides a clearer image of the glove strap section. Experimentation showed that these segments were necessary to model the glove strap region correctly. This is explained shortly.

TABLE 5.1: Number of BUs by which each segment was extruded.

Segment	Extrusion (BUs)
Glove strap segment 1	0.005
Glove strap segment 2	0.060
Glove strap segment 3	0.005
Pre-palm segment	0.006
Palm	0.630

Before the modelling process continued, a level 1 Catmull-Clark subdivision surfacing modifier was added to the model. This modifier allows the modeller to view a preview of the smoothed model and increase the subdivision surfacing level without modifying the original model. The vertices on the underlying model then act as control points that parametrically deform the smoothed model preview. The modifier can later be permanently applied to the model in which case the underlying model is converted into the smoothed model by increasing the number of vertices and transforming them as required.

We decided not to apply the modifier to the model during the modelling process. The few vertices on the underlying model provided sufficient control over the modelling process. The preview facility was used to monitor the outcome of modifications made to the model. Figure 5.3 depicts the model and the smoothed preview after applying the subdivision surfacing modifier.

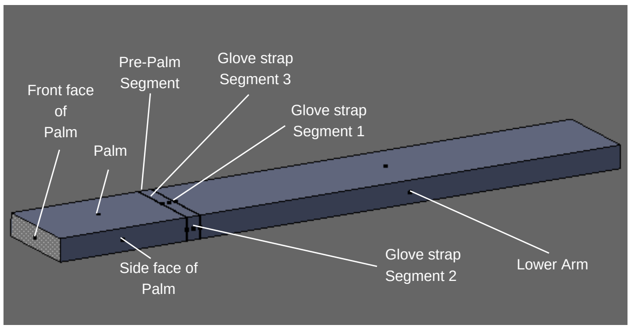


FIGURE 5.1: The arm model after extrusion of features up to and including the palm.

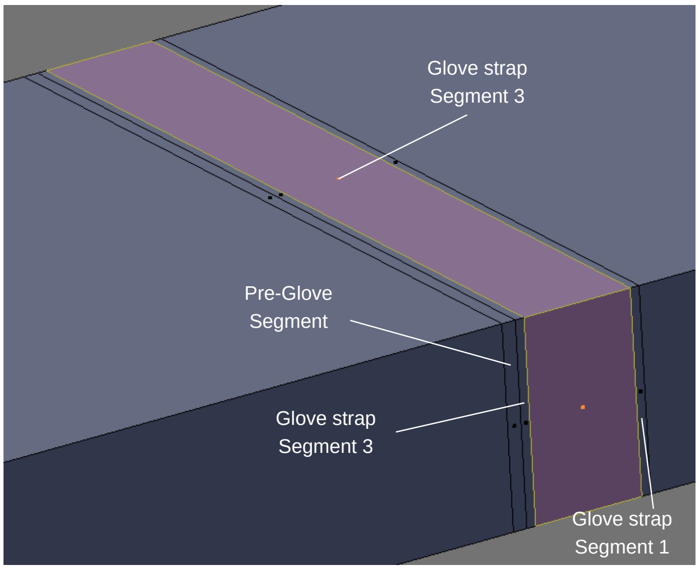


FIGURE 5.2: A close-up of the glove strap segments.

Figure 5.4 is a close-up of the model with background occlusion disabled to show five faces, A, B, C, D and E of the model. Henceforth, we refer to the faces of each segment in terms of these five faces.

The glove strap and the palm were modelled to give them a more swollen glove-like look. The palm was also modelled to a size slightly wider and thicker than the arm. Faces A and C of segment 2 of the glove strap were shifted outwards by 0.03 BUs while faces B and D were shifted outwards by 0.04 BUs. Faces A and C of the palm were shifted outwards by 0.09 BUs while faces B and D were shifted outwards by 0.03 BUs. The difference in units by which these faces were shifted outwards was in line with the difference in width and thickness of the arm. The four vertices joining the pre-palm segment and the palm were then shifted towards and aligned with the vertices of segment 3 of the glove strap. This created a clear buffer between the glove strap region and the palm. The result of these processes is depicted in Figure 5.5a. For further clarity, Figure 5.5b shows a solid version of the smoothed preview model.

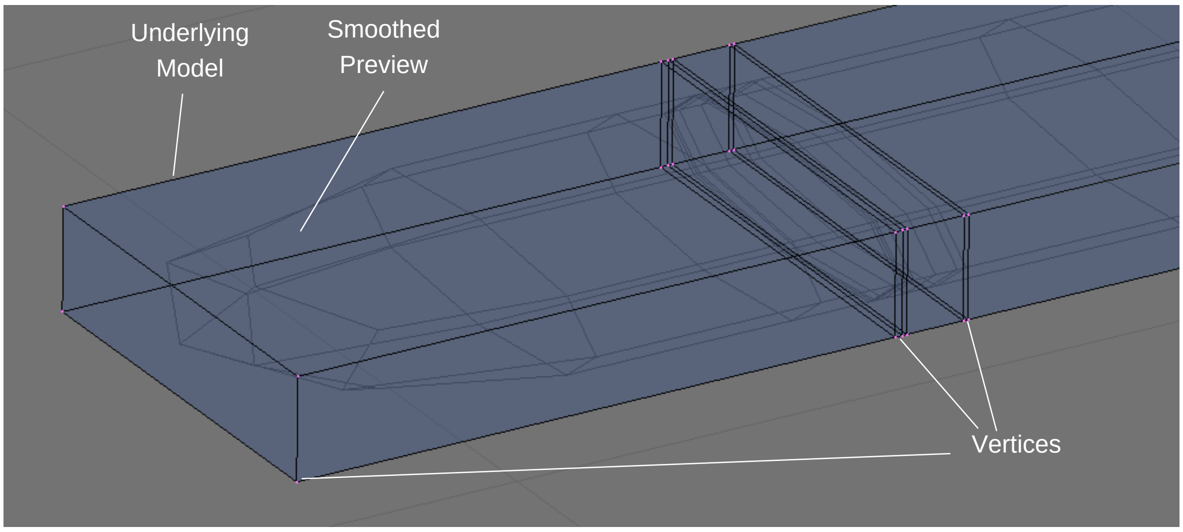


Figure 5.3: The model and smoothed preview with Catmull-Clark subdivision surfacing level 1 applied.

A box modelling technique was then used to model the fingers. Face E depicted in Figure 5.4 was subdivided into four equally-sized faces using the subdivision tool, one square for each of the four fingers. The fingers were modelled one at a time sequentially starting with the index finger and ending with the pinky. For each finger, the square on face E was extruded outwards 3 times, one segment for a pre-finger segment, and two segments for proximal and distal phalanges. It was decided to leave out the middle phalanges since two phalanges provided a sufficient amount of detail on the finger. Experimentation showed that the pre-finger segment was necessary to model the joints between fingers and the palm correctly. The number of BUs by which the segments of each finger were extruded is summarized in Table 5.2.

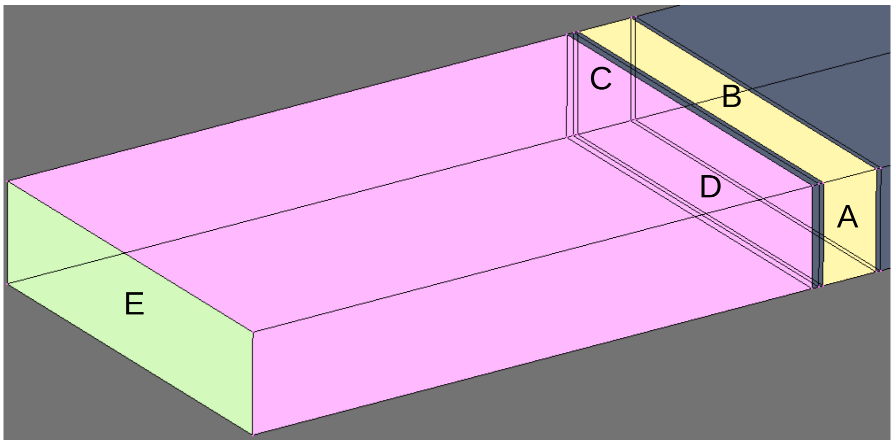


FIGURE 5.4: A close-up of the model with background occlusion disabled to show five faces, A, B, C, D and E of the model.

TABLE 5.2: Number of BUs by which segments in each finger were extruded.

Finger	**Extrusion (BUs)**		
	Pre-Finger Segment	**Proximal Phalange**	**Distal Phalange**
Index	0.03	0.21	0.31
Middle	0.03	0.24	0.38
Ring	0.03	0.21	0.31
Pinky	0.03	0.12	0.21
Thumb	0.03	0.14	0.20

Each finger was then modelled such that the joint was collapsed and the finger progressively grew in size up to the finger tips. The four edges between the pre-finger segment and the proximal phalange were selected and scaled down by a factor of 0.44 in the x and y directions. The vertices on these edges were shifted towards the palm perpendicularly by 0.05 BUs. The face on the tip of the finger was then selected and scaled up by a factor of 1.6 in the x and y directions. At this point, all four fingers had been modelled but overlapped.

This was resolved by rotating and shifting fingers. This was done by rotating all vertices in the finger, pivoted on the joint between the finger and the palm, and shifting the vertices as required. Table 5.3 summarizes the number of BUs by which each finger was shifted and the angle by which it was rotated.

The thumb was then modelled. Face A on the palm segment was subdivided into three faces F, G and H, as shown in Figure 5.6. The two edges between faces F and G, and G and H were shifted along the z-axis until the three faces were of the lengths seen in Figure 5.6. Face G was extruded outwards 3 times to form a pre-finger segment, a proximal phalange and a distal phalange. The lengths of these segments are summarized in Table 5.2. The edges between the pre-finger segment and the proximal phalange were then scaled down by a factor of 0.70 in the z-direction and 0.43 in the y-direction. They

Table 5.3: Rotations and shifts applied to fingers.

Finger	**Rotation (°)**	**Shift (BUs)**		
		X-axis	**Y-axis**	**Z-axis**
Index	-12	0.012	0	0.018
Middle	0	0	0	0
Ring	11	0	0	0
Pinky	19	-0.06	0	0.006
Thumb	32	0.026	0	-0.01

were shifted towards the palm perpendicularly by 0.025 BUs. The face on the tip of the thumb was scaled up by factor of 2.2 in the z direction. The whole thumb was rotated and shifted to a location and angle that looked plausible for a thumb. The shift and rotation applied are summarized in Table 5.3.

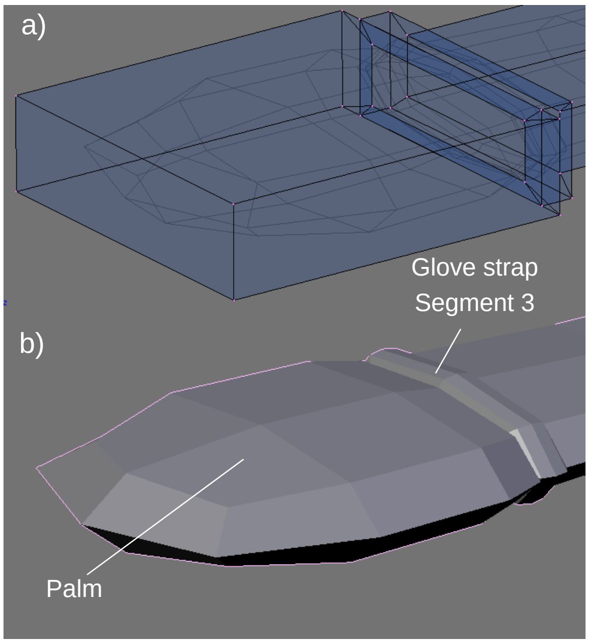


Figure 5.5: (a) The resultant model and b) The resultant smoothed preview, after modelling the glove strap and palm.

The edges of the face on the tip of the forearm were then creased by 0.76 BUs to provide a more rounded look, as seen in Figures 5.7a and 5.7b respectively. At this point, we attempted to determine the highest Catmull-Clark subdivision surfacing level that was renderable on the mobile phone. A mirror modifier was added and applied to the arm model to obtain an exact copy of the right arm inverted in the x direction. This formed the left arm. The subsurface division level was increased to 2 and the modifier was permanently applied to the model to obtain the smoothed model. The model was exported and tested with TestModel. It was found that the importation failed. It was decided to keep the subsurface division level at 1.

The subsurface division level was set back to 1 and the modifier permanently applied to the model. A white material was applied to the faces of the glove strap, palm and fingers. A skin-coloured material, with red, green and blue values of 240, 209 and 163 respectively, was applied to the faces of the forearm. Finally, The arms were placed 2 BUs away from either side of the z-axis and 4.6 BUs above it, with no offset from the y-axis, as measured from the vertex at the tip of the forearms. The completed arms are shown in Figure 5.8.

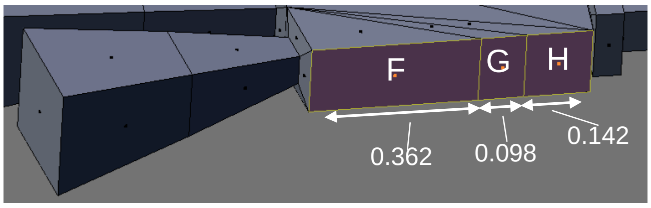


FIGURE 5.6: The completed fingers.

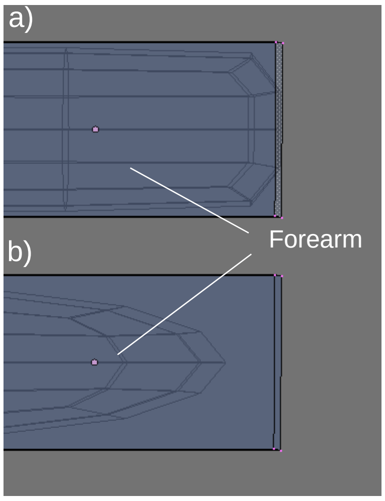


FIGURE 5.7: The tip of the forearm (a) with the crease modified (b) with the crease unmodified.

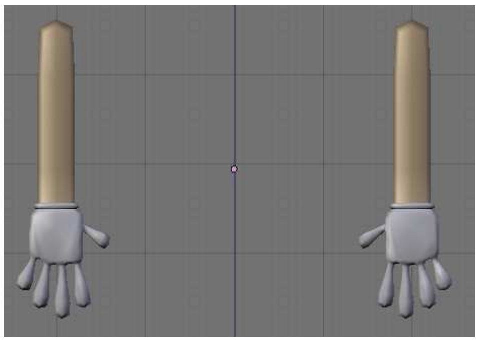


FIGURE 5.8: The completed arms.

At this stage, the model was exported and imported into the application TestModel. It was found that the animation of the model took place at approximately 7 frames per second (fps).

We then acquired the head of *Man* for use with *Phlank*. We imported the entire *Man* model into the Blender project and all but its head and neck was then deleted. At this point the model was exported and imported into TestModel to test its performance. It was found that the model could not be imported into TestModel. An error was reported that indicated that the phone had run out of memory. This meant that the model had too many vertices and was too detailed. We resolved to reduce the detail on the head. Reduction of features was carried out until the model successfully imported into and rendered in TestModel. Features were systematically removed in order of their importance which was determined as explained shortly. After each removal, the model was tested in TestModel to test for importation success.

The face itself was considered the most important feature on the head. All other features on the head were considered to be of varied importance. Features that were not visible during rendering were regarded as the least important features. The importance of all other features was graded according to their distance from the face. Features closer to the face provide more detail to the face and were graded as more important that those farther away.

Table 5.4 lists the features removed in ascending order of importance. The last entry in this table was the final feature removed before the model imported into TestModel successfully. The frame rate of animation reduced to approximately 4 fps. The small reduction in frame rate was justified by the increase in detail of the Avatar. As mentioned previously, increasing the quality of frames with reductions in frame rate provides enhanced SL intelligibility. The head was then positioned at 7.6 BUs above the z-axis and -0.6 BUs from the y-axis, with no offset from the x-axis, all measured with respect to the highest vertex in the center of the forehead. The completed model is depicted in Figure 5.9. Table 5.5 summarizes the number of vertices, edges and faces on the model.

5.1.2 Parameterization

The model was parameterized using a skeleton as in Section 4.1.2.1 of Chapter 4. We started by adding the bones of the right arm. An armature was added and positioned such that its root was at x-, y- and z-coordinates -2.0, -2.0 and 6.3 respectively, and its tip was at x-, y- and z-coordinates -2.0, -2.0 and 4.6 respectively. Its root was positioned at the approximate location of the shoulder, had there been one, and its tip, at the tip of the forearm, where an elbow would have been. Other armatures were then added to form a skeleton for the arm. This was done by estimating the locations of joints. Table 5.6 summarizes all the bones in the skeleton of the right arm, the names they were assigned, and the position of their tips relative to their parent bone. Figure 5.10 depicts

Table 5.4: Features removed from *Phlank* in ascending order of importance until the model was importable and renderable on the mobile phone.

Order	Feature
1	Tongue, teeth and all parts of the inner mouth and inner ears
2	Back of the head
3	Back of the eyes leaving only two sphere segments slightly wider than the space between the eyelids
4	Back of the neck
5	Back of the ears
6	Side of the head
7	Remaining parts of the neck up to the portion just under the chin
8	Top of the head
9	Remaining parts of the ears
10	Forehead up to three vertex rows above the eyebrows
11	Sides of the face leaving only the cheeks

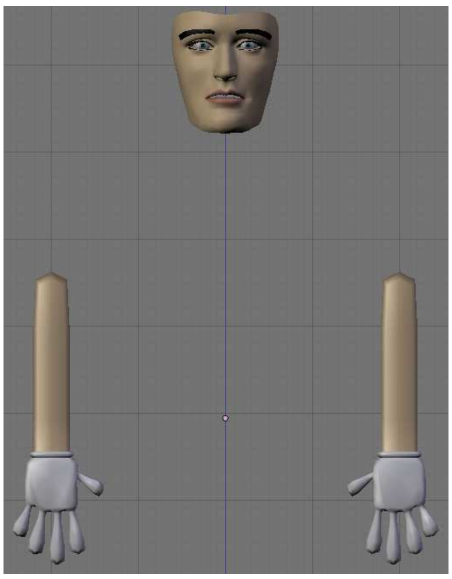


Figure 5.9: The completed *Phlank* model.

Table 5.5: The total number of vertices, edges and faces on the completed *Phlank* model.

Subset	Vertices	Edges	Faces
Head	1073	2071	996
Arms	864	1720	860
Total	**1937**	**3791**	**1856**

the hierarchical structure of the tree of bones in a format similar to that of H-Anim. It indicates the parent-child relationships between armatures.

The bones of the right arm were then duplicated and rotated by 180° on the z-axis. The

Table 5.6: The bones in the right arm, their assigned names, and the position of their tips relative to their parent bone.

Bone	**Assigned Name**	**Relative Position (BUs)**		
		X-axis	**Y-axis**	**Z-axis**
Upper Arm	r_upper	-	-	-
Forearm	r_fore	0	0	-2.0
Wrist	r_wrist	0	0	-0.2
Pinky Metacarpal	r_h1	-0.246	0	-0.478
Ring Finger Metacarpal	r_h2	-0.095	0	-0.527
Middle Finger Metacarpal	r_h3	0.069	0	-0.535
Index Finger Metacarpal	r_h4	0.240	0	-0.528
Thumb Metacarpal	r_h5	0.309	0	-0.129
Pinky Proximal Phalange	r_pinky1	-0.053	0	-0.130
Ring Finger Proximal Phalange	r_ring1	-0.041	0	-0.207
Middle Finger Proximal Phalange	r_middle1	-0.004	0	-0.242
Index Finger Proximal Phalange	r_index1	0.034	0	-0.199
Thumb Proximal Phalange	r_thumb1	0.129	0	-0.076
Pinky Distal Phalange	r_pinky2	-0.063	0	-0.158
Ring Finger Distal Phalange	r_ring2	-0.050	0	-0.248
Middle Finger Distal Phalange	r_middle2	0.007	0	-0.303
Index Finger Distal Phalange	r_index2	0.034	0	-0.226
Thumb Distal Phalange	r_thumb2	0.113	0	-0.088

new bones were then placed onto the left arm, 4 BUs away from the bones of the right arm in the x-direction and with the same z- and y-offset as the bones of the right arm.

The entire skeleton was then attached to the model automatically by means of the built-in algorithm by Baran and Popović. Figure 5.11 depicts the model with its attached skeleton.

```
r_upper
  r_fore
    r_wrist
      r_h1
        r_pinky1
          r_pinky2
      r_h2
        r_ringfinger1
          r_ringfinger2
      r_h3
        r_middlefinger1
          r_middlefinger2
      r_h4
        r_indexfinger1
          r_indexfinger2
      r_h5
        r_thumb1
          r_thumb2
```

FIGURE 5.10: Hierarchical structure of *Phlank*'s skeleton.

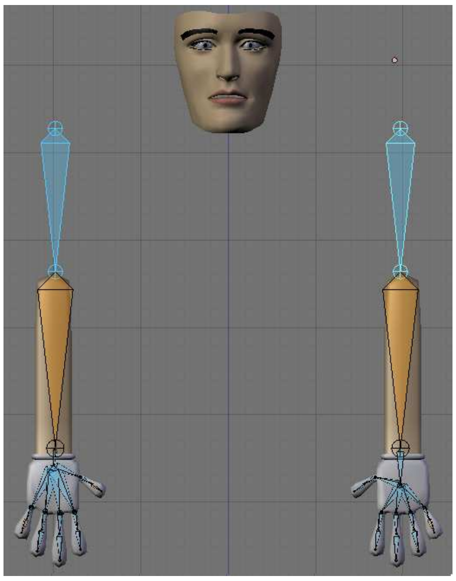


FIGURE 5.11: The *Phlank* model with a completed skeleton.

5.2 Animation and Exportation of *Phlank* and *Man*

5.2.1 Animation

For the purpose of this project, we selected 16 short common phrases from a phrase book, most of which consisted of single words. Table 5.7 lists these phrases. The corresponding SASL of these phrases was obtained by recording SASL speakers. We enlisted the help of the Dominican School for the Deaf in Wynberg, Cape Town. Five profoundly deaf students whose first language was SASL were requested to sign each of the phrases under the supervision of a hearing teacher with expert knowledge of SASL. Each phrase was recorded and labelled. We obtained ethics clearance and each subject was provided a consent form whose contents were explained to the subject by the hearing teacher.

The student was informed of the purpose of the research as well as of the fact that participation was voluntary and it was possible to withdraw from the exercise at any time.

TABLE 5.7: List of SASL phrases recorded and animated.

No	Phrase	No	Phrase
1	Bus	9	Medicine
2	Doctor	10	Restaurant
3	Good Evening	11	Right
4	Hello	12	Sick
5	Help Me	13	Soccer
6	Help You	14	South Africa
7	How	15	Toilet
8	Left	16	Water

Using the SASL videos as a basis and with careful attention to detail, both *Phlank* and *Man* were animated using a keyframing technique. Two sets of animations were produced for *Man*. The first set included the non-manual gestures observed in the SASL videos while the second set excluded them. Two techniques were used in keyframing each phrase. The first technique involves identifying frames in the SASL video at which the hands make extreme direction or speed changes. These frames are then noted as keyframes. The second technique was proposed by Terra and Metoyer in [130]. They proposed that making a clear distinction between keyframe values and keyframe timing simplifies the animation process.

Our animation methodology took place in the following steps:

1. Identify frames in the SASL video where the hands make extreme direction or speed changes.
2. For each of these frames, pose the model in the same pose as in the frame.
3. Add the pose as a keyframe in the animation.
4. Once all keyframes have been added, shift each keyframe in the timeline of the action editor to the correct time.

5.2.2 Exportation

Exportation of the *Phlank* model was carried out by means of the M3G exporter plug-in. This was a simple process of invoking the M3G exporter script, setting the required parameters and initiating the exportation process. The script produces an M3G file of the model that can be imported onto the mobile phone using the Mobile 3D Graphics library in Java ME and subsequently played back as a 3D model.

Parameters of the M3G exporter that we made use of are summarized in Table 5.8.

TABLE 5.8: Key parameters of the M3G exporter tool.

Parameter	**Description**
All Armature Actions	Sets whether or not to export all actions in the animation database of the skeleton
Smooth Shading	Sets whether or not to use smooth shading in PolygonMode
Lighting	Sets whether or not the lights in the scene should be exported with the model

All three parameters were set to the “On” position.

It was initially hoped that it would be possible to export *Man* using the same method of exportation used for *Phlank*. It was discovered that this was not possible due to the extreme large size of the model. A process of systematically minimizing *Man* was then carried out by removing the least critical features. As was the case with the minimization of the face of *Phlank*, explained in a previous section, the importance of features was determined by their necessity in sign language visualization. The face, forearms and hands were considered the most important features. All other features were assigned an importance based on their visibility during rendering followed by their distance from the most important features. In this case, the head could not be minimized to the same extent as with the face of *Phlank* since this would lead to a dislocation of its facial parameterization, thereby invalidating a key justification for doing so. After each removal, an attempt at exportation and importation into TestModel was made. Reduction of the model took place in the order summarized in Table 5.9.

TABLE 5.9: Features removed from *Man* in ascending order of importance.

Order	**Feature**
1	Removal of the lower body.
2	Removal of the abdomen region.
3	Removal of the faces on the body that are not visible during animation, that is, the back.
4	Removal of the faces on the head that are not visible during animation, that is, the back of the head.
5	Removal of all faces excepting the arms and the head.
6	Removal of the upper arms leaving the forearms and hands.

Exportation was found to be unsuccessful at every stage. After the removal of the upper arms, it was decided that no additional features could be removed such that *Man* could still be considered useful for detailed SL rendering. We resolved to export the animations and render them on the mobile phone as video. They were exported to the MPEG-4

Part 14 (MP4) format and imported into the Mobile Media API of Java ME and played as a video file.

5.3 Summary

We have presented the methodology used in the creation of our low-detail Avatar *Phlank*. We explained our implementation of each step of the SL Avatar creation methodology. We also explained the method of animating and exporting the Avatars *Phlank* and *Man*. We showed that it was not possible to export *Man* as a 3D model that can be rendered on a mobile phone.

Chapter 6

Implementation of a Prototype Mobile Framework

In Chapter 2, we established the need for a mobile-based communication tool for the deaf. In addition to developing the mobile Avatar *Phlank*, we developed and implemented a prototype mobile phone-based machine translation system that we called the iSign Mobile Framework into which we integrated our Avatar. This implementation is not only the first tangible implementation of a unified translation system for the SASL group but, as we have mentioned in Chapter 2, is the first unified implementation that can carry out translation from spoken language to sign language **and** sign language to spoken language, that we aware of. In addition to this, it has been implemented on a mobile phone framework, the most powerful application of such a system. Therefore, our work is a pioneer in the field.

The iSign Mobile Framework is a realization of the objective of the SASL project at the University of the Western Cape [28], to adapt the conceptual design of the Machine Translation (MT) system of the same project onto a mobile framework. In this section, we first provide a conceptual overview of the generic Machine Translation system of the SASL project followed by an enumeration of the components of the system that currently exist, having been the subjects of previous projects. We then propose an adaptation of the generic architecture into a mobile framework architecture. We explain our implementation of a prototype of the iSign Mobile Framework architecture as a proof of concept.

6.1 The SASL Machine Translation System

6.1.1 Conceptual Overview

The SASL project of the SASL group at the University of the Western Cape is in the process of designing and implementing a MT system [28] that automates the translation process between SASL and English using unobtrusive hardware. Within this system, two separate functions are catered for; converting sign language video into English and converting English into Sign Language video. The implementation of these two functions in the translation system allows for two-way communication between deaf and hearing users. Each function can be subdivided into four sequential steps; capturing input, recognizing the captured input, translating the input into the target language, and rendering the target language. Figure 6.1 depicts the conceptual structure of the SASL MT system.

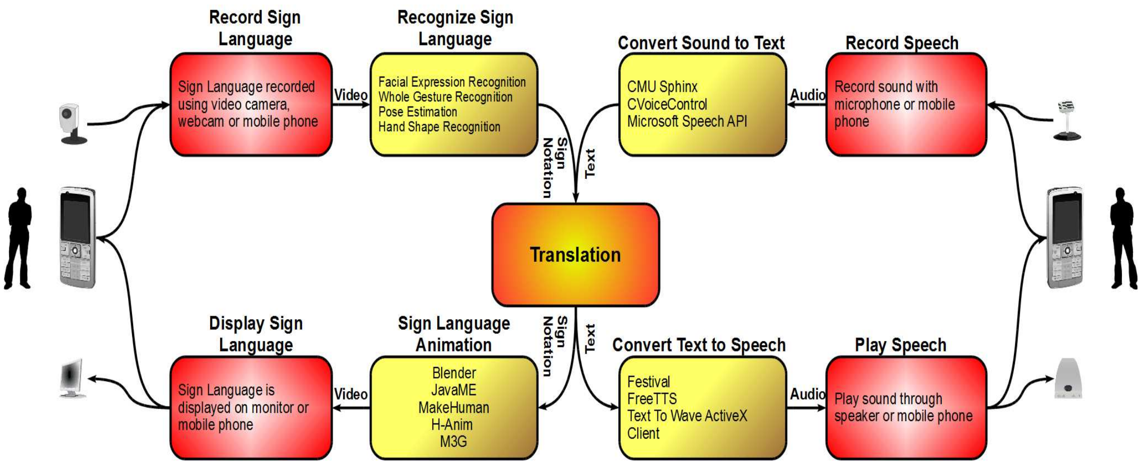


FIGURE 6.1: Conceptual overview of the SASL Machine Translation system [28].

With reference to Figure 6.1, the conversion of SASL into English is ideally carried out in the following four steps:

1. **Capture**: Video of a person gesturing SASL is recorded. This is carried out by means of a webcam or mobile phone camera.

2. **Recognition**: Computer vision techniques are used to extract semantic information from the input video. The information that is extracted includes both manual gestures, such as hand motions and shapes, and non-manual gestures, such as body, neck and head motion and facial expressions. The information extracted is transcribed using a machine-readable form of a SL transcription notation such as Sign Writing Markup Language (SWML) [109] and Systems Biology Markup Language (SBML) [126] for SignWriting [125] or ASCII-Stokoe [80] for Stokoe[117].

3. **Translation**: The transcribed SASL is translated into English text.
4. **Conversion**: The English text is converted into English audio using a text-to-speech synthesizer.
5. **Rendering**: The English audio is rendered through the speaker. This can either be on a desktop computer or on a mobile phone.

The conversion of English into SASL is ideally carried out in the following four steps:

1. **Capture**: The audio of a person speaking English is recorded. This is carried out by means of an external or internal microphone on a desktop computer or the microphone on the mobile phone.
2. **Recognition**: Speech recognition is carried out on the audio, converting it to English text.
3. **Translation**: The English text is translated into transcribed SASL.
4. **Conversion**: The transcribed SASL is used to synthesize an Avatar signing SASL.
5. **Rendering**: The Avatar is rendered. This may be done on a desktop computer or mobile phone screen.

6.1.2 Existing Components

To-date, the SASL group has focused on developing the various components of the MT system described in the previous subsection independently. Thus far, these components have all been built for desktop computers.

The recognition of captured SASL has been the subject of many projects of the SASL group. These systems were explained in Section 2.2.3.2 of Chapter 2. Naidoo [88] and Rajah [99] created systems that can recognize SASL from video input. Segers [106] created a system that can recognize hand-shapes from video input. Whitehill's system [144] can recognize and classify facial expressions from video input. However, these systems do not currently produce computer-readable SL transcription but English text instead.

The SASL group currently focuses on recognition and animation and is yet to focus on translating between SASL and English.

The conversion of English text into English audio is an active field of research with projects such as Festival [39], FreeTTS [42] and eSpeak [36]. These systems can be integrated into the SASL MT system proposed.

The conversion of English audio into English text is also an active field of research with projects such as Simon [110], CMU Sphinx [26] and VoxForge [139]. These systems are also open-source and can be adopted in the SASL MT system.

As explained in Chapter 4, Van Wyk's system [146] is capable of rendering high quality SASL signs from SBML input.

6.2 The iSign Mobile Framework Architecture

The SASL MT system explained in the previous subsection was adapted into a mobile framework – the iSign Mobile Framework. Mobile phones are still far less powerful than desktop computers. They lack memory and processing power. This was a major consideration in designing the iSign Mobile Framework. We aimed to use existing systems mentioned in subsection 6.1.2. However, these systems rely on the capabilities of desktop computers. It was necessary to redistribute the components of the generic SASL MT system architecture in a way that would enable the use of these components within the mobile framework.

Our architecture proposes that all capturing and rendering processes be carried out on mobile phones since these are within the capabilities of the mobile phone that we have selected. A mobile application on the phone facilitates all such processes. We then propose that all systems carrying out recognition and translation processes, as well as the conversion of English text into English audio, be placed on a server referred to as the translation server. The mobile phone can then access these systems as services on the server using a web-based client/server architecture. The conversion of transcribed SASL into an Avatar may be placed on the mobile phone if the Avatar is compact enough to be rendered as a 3D model on the phone and the phone supports 3D rendering. If not, this process is also carried out on the translation server.

A web service on the server mediates between the mobile phone and the services on the server by means of a communication protocol. Figure 6.2a depicts the generic architecture of the iSign Mobile Framework and the process of translating SASL into English audio. Figure 6.2b depicts the same architecture and the process of translating English audio into SASL.

The Internet cloud depicted in Figure 6.2 is an abstraction of the underlying Internet infrastructure that necessarily consists of Edge, GPRS, WiFi or 3G technologies. These technologies are required to establish wireless connectivity between the phone and the

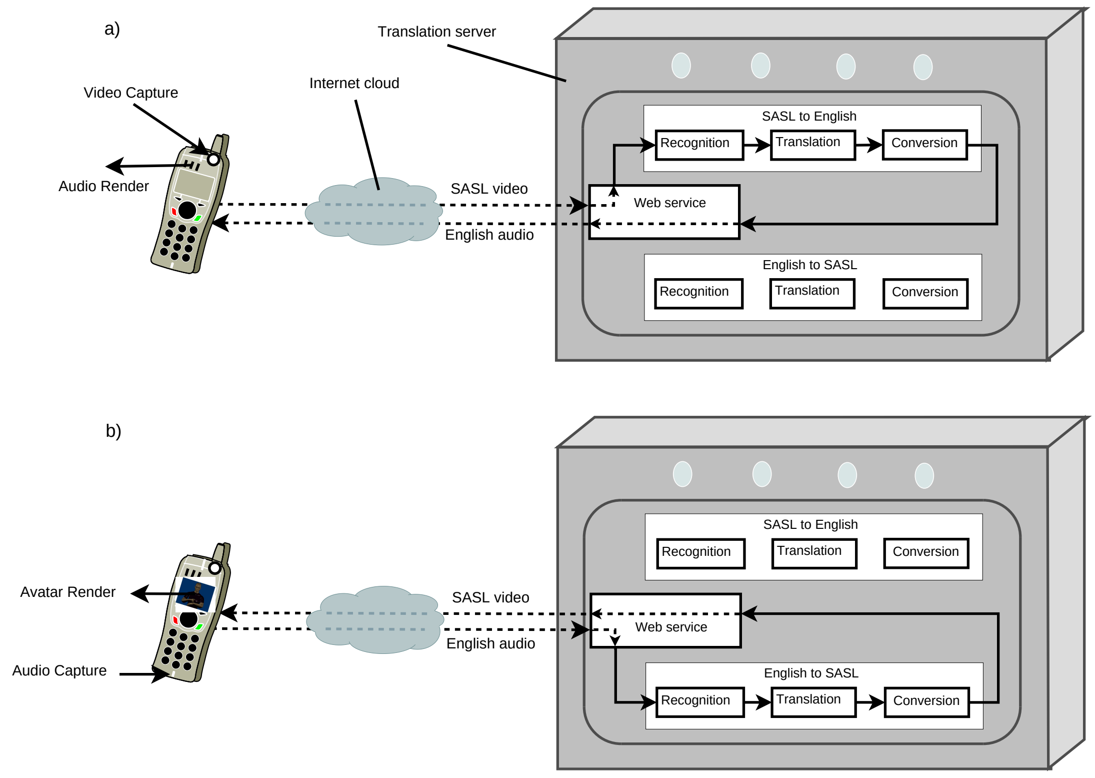


Figure 6.2: Overview of the iSign Mobile Framework showing the process of a) SASL to English audio translation and b) English audio to SASL translation.

network. Within these figures, the "conversion" process has been included in the translation from English audio to SASL but this need not be the case if the Avatar can be rendered on the phone, as mentioned previously.

Table 6.1 summarizes the key components of the iSign Mobile Framework. It is important to note that this architecture establishes a well-defined modularized approach to the SASL/English MT problem. Each of the components of this architecture may be developed, improved and changed independently of all other components as long as the communication protocol is adhered to. This is particularly suitable since many of these components are actively being developed in various SASL project research efforts.

6.3 Implementation of the iSign Mobile Framework

The implementation of the iSign Mobile Framework involved putting prototype components in place for each of the components mentioned in Table 6.1. The components configured included the interpretation services for SASL to English and English to SASL, the web-service, the mobile application and a prototype communication protocol.

Table 6.1: Key components of the iSign Mobile Framework.

Component	**Description**
Mobile application	Facilitates video and audio capturing and rendering, rendering of 3D models and communication with the translation server
SASL to English interpretation services	Carry out interpretation from SASL video to English audio
English to SASL interpretation services	Carry out interpretation from English audio to SASL video or Avatar
Web service	Mediates between the mobile phone application and the interpretation services
Communication protocol	Defines a standard mode of communication between other components

6.3.1 The Communication Protocol

We developed a reduced communication protocol for the purposes of the prototype. It should be noted ahead of time that this protocol, like the other components in the prototype implementation, may be subject to improvement or replacement by a comprehensive standard protocol such as the Session Initiation Protocol (SIP).

The protocol defines five communicative tasks utilized in this translation system framework. These are:

1. **Login**: The mobile phones in the framework are required to login to the web service with a pre-stored username and password. A session is initiated.
2. **Connecting to a phone**: The user of the mobile phone selects a target phone logged into the system to which interpreted messages will be sent.
3. **Uploading a job**: A mobile phone may send 'jobs' in the form of SASL video or English audio to the web-service for interpretation.
4. **Processing the job**: An uploaded job is retrieved and processed by the relevant interpretation services.
5. **Retrieving the job**: The resultant interpreted message is retrieved by the phone for rendering.

As mentioned, the protocol allows a phone to connect to other phones for communication. It also supports a 'Translator' mode in which the phone itself is the target phone. In this case, the phone acts as a stand-alone interpreter, capturing both SASL and English jobs and retrieving processed jobs and rendering them. Figures 6.2a and 6.2b depict translator mode-type operation. We limit the current discussion to the 'Translator' mode.

The web service is at the core of the protocol. Each of the tasks mentioned involves a set of defined communication exchanges between the web service – the server – and either the interpretation services or the mobile application – the client. Each communication exchange is defined by a signal and a set of accompanying parameters that are sent to the web service by the client. The signal and parameters invoke a specific reaction on the web service. The web service may also return a value depending on the task at hand. Table 6.2 lists the signals in the communication protocol and provides a description of each signal. For more information, including the parameters and return values of each signal, the reader is referred to Tables B.1, B.2, B.3, B.4 and B.5 of Appendix B.

Table 6.2: Signals in the communication protocol and their descriptions.

Signal	**Description**
login	Sends the pre-stored phone number and pin. If login is successful, a session id is returned. If not the phrase "Error" is returned.
contacts	Retrieves a list of phones registered on the framework that are currently logged in and available.
connect	Attempts to connect to a phone. Return values correspond to the target phone being unavailable and waiting for it to respond to the connect request.
setmode	Sets the current capture and render mode to either deaf (SASL) or hearing (English) mode. Returns "Error" if the connection was interrupted.
vidin	Sends a SASL video message to be processed and sent to the target phone. Return values correspond to success and failure of the upload respectively.
wavin	Sends a English audio message to be processed and sent to the target phone. Return values correspond to success and failure of the upload respectively.
getvidjob	Queries for any unprocessed video jobs. Returns "None" if there are no such jobs or information on the file to be retrieved.
getwavjob	Queries for any unprocessed audio jobs. Returns "None" if there are no such jobs or information on the file to be retrieved.
vidjobdone	Notification of successful completion of SASL to English interpretation processing.
wavjobdone	Notification of successful completion of English to SASL interpretation processing.
vidjobfailed	Notification of unsuccessful completion of SASL to English interpretation processing.
wavjobfailed	Notification of unsuccessful completion of English to SASL interpretation processing.
querymsg	Queries for any interpreted messages directed at this phone. Returns "None" or a URL to the file to be retrieved.

Specific signals in Table 6.2 are sent to the web-service at regular intervals in order to poll it for a response, based on which an action is taken. These are the *contacts*, *querymsg*, *getvidjob* and *getwavjob* signals. The polling interval was set to 8 seconds since this provided a balance between polling latency and bandwidth usage. Figure 6.3 is a sequence diagram containing the sequence of signals sent for the five tasks. It is to be noted that the polling tasks are shown separately with a 'polling' box around them.

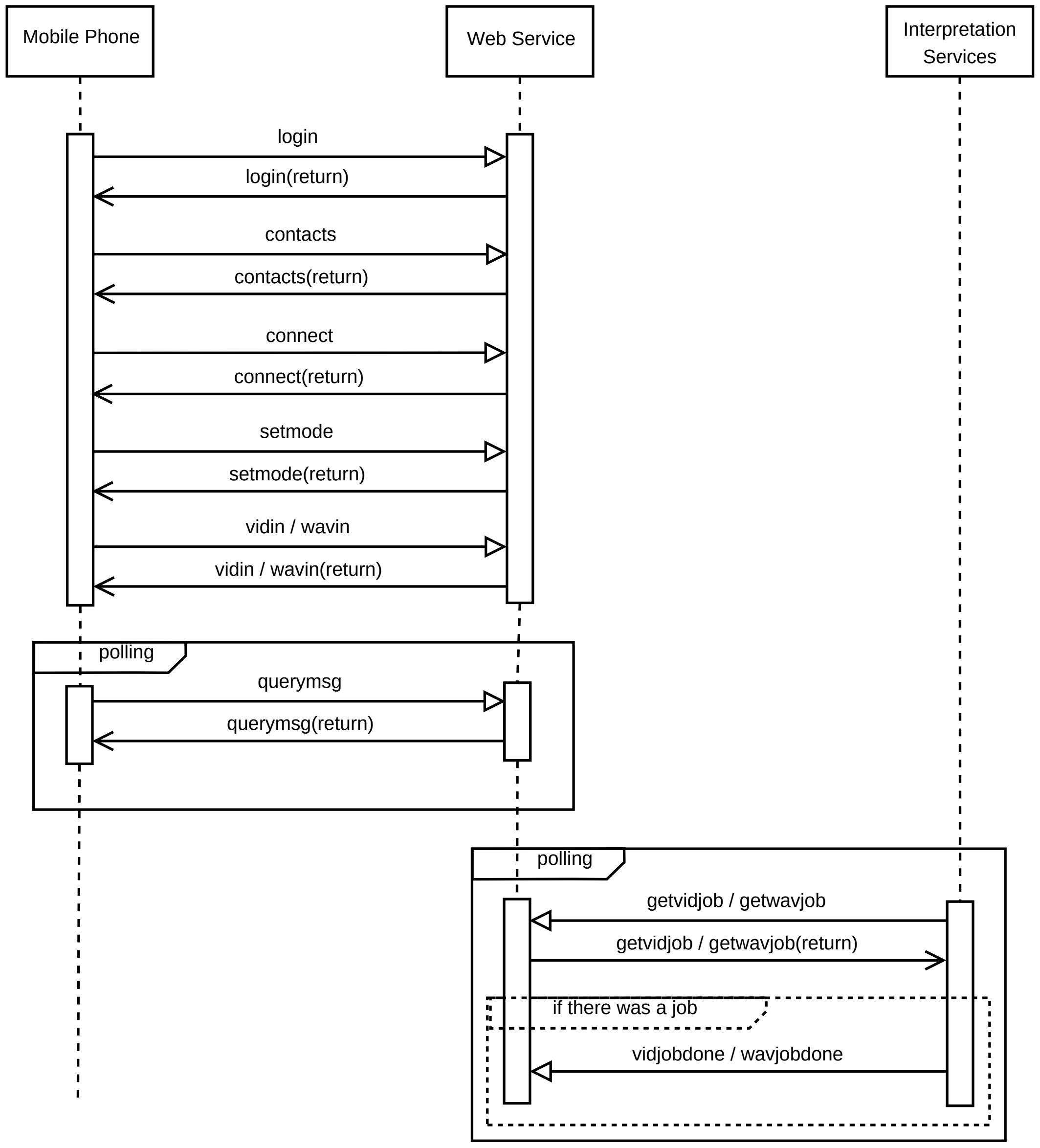


FIGURE 6.3: Sequence diagram of communication exchange signals.

6.3.2 The Web Service

The web service is implemented on an Apache HTTP server installation. It takes the form of a PHP script that can be passed signals of the aforementioned protocol and their parameters in the form of HTTP GET and POST parameters. The signal is specified by passing the script a GET parameter called ‘act’ and its value is set to the signal. The parameters of the signal are passed in as either GET or POST parameters named as in Table 6.2.

The web service makes use of a MySQL database to store information pertaining to phones registered on the service, login sessions and jobs. It contains 4 tables: USER, CONTACTS, VIDEOIN and WAVEIN. Table 6.3 provides a brief description of each of these tables. For more information, the reader is referred to Tables B.6, B.7, B.8 and B.9 of Appendix B which are data dictionaries for the tables in the database.

Table 6.3: Description of the function of each table in the iSign web service database.

Table	**Description**
USER	Stores information pertaining to mobile phones registered on the service.
CONTACTS	Stores a list of contacts that have been added and can be connected to, for each phone.
VIDEOIN	Keeps track of SASL video jobs sent to the web service for translation.
WAVEIN	Keeps track of English audio jobs sent to the web service for translation.

6.3.3 The Mobile Application

We developed a mobile application in Java ME that we called iSign. This application facilitates all processes involving capturing, rendering and network communication. It further adheres to other requirements of the communication protocol such as facilitating a login process, displaying a list of contacts, allowing the selection of a contact to be connected to and allowing the selection of a capture and render mode. Figure 6.4 depicts the Sony Ericsson C905 that we used. We have labelled three specific buttons, button 1, button 2 and button 3, on the phone for reference in the subsections that follow.

The subsections that follow provide screenshots and a brief description of the interface of the application that facilitates the processes mentioned.

6.3.3.1 The *Login* Screen

Figure 6.5a shows the initial state of the *Login* screen. It is the default iSign screen. The user may specify the URL of an iSign Translation Server. This is done by pressing button 1 which invokes the *Settings* screen depicted in Figure 6.5b. The user may specify their login details, consisting of a pre-registered mobile phone number and a pin as seen in Figure 6.5c and pressing the ‘Login’ button. The current implementation

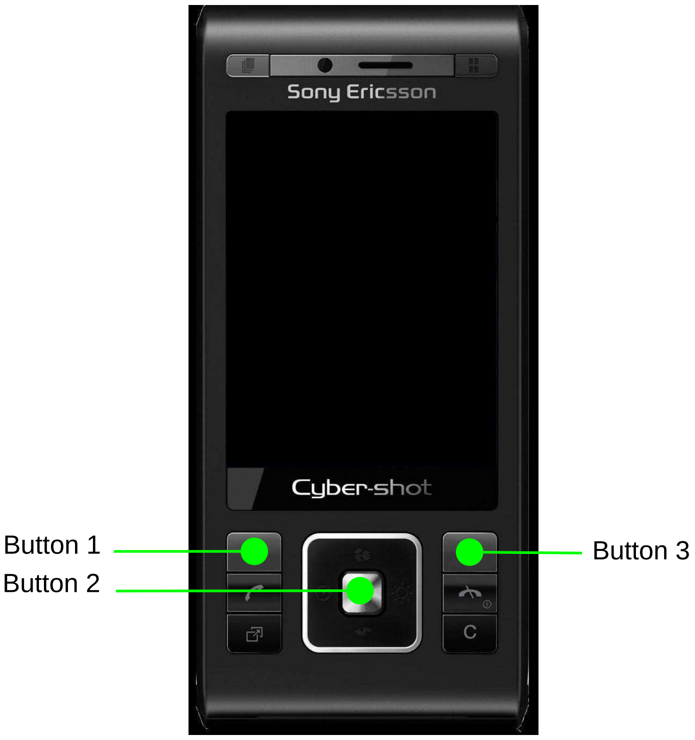


FIGURE 6.4: The Sony Ericsson C905.

does not include a registration process but this feature can be incorporated. iSign sends the information entered to the iSign Translation Server for verification and displays the screen depicted in Figure 6.5d.

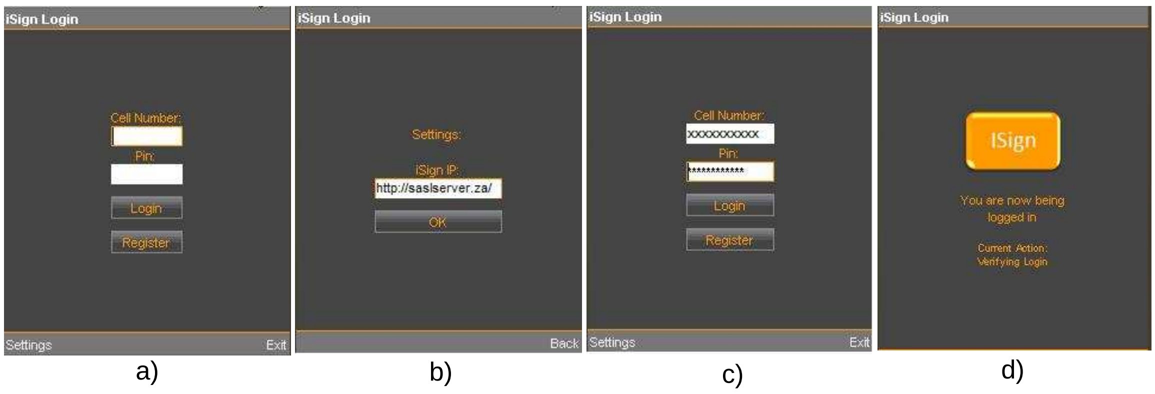


FIGURE 6.5: (a) The initial *Login* screen. (b) *Settings* screen for specifying the Translation Server URL. (c) Login details filled in. (d) Screen shown while sending information to the Translation Server and verifying login.

6.3.3.2 The *Contacts* Screen

Figure 6.6a shows the initial *Contacts* screen. The main feature of this screen is the contacts list that displays a list of all contacts currently added by the user of this mobile phone, as well as an entry entitled ‘Translator’. Each contact entry has a status light that may be grey, green or orange corresponding to not logged in, available and not available, respectively. The list may be navigated to select any of its entries. Figure 6.6b shows the *Translator* entry selected. The user presses button 2 to connect to that contact. In this case, the user initiates the Translator mode. Figure 6.6c is displayed while iSign sends the required signal and parameters to the Translation Server.

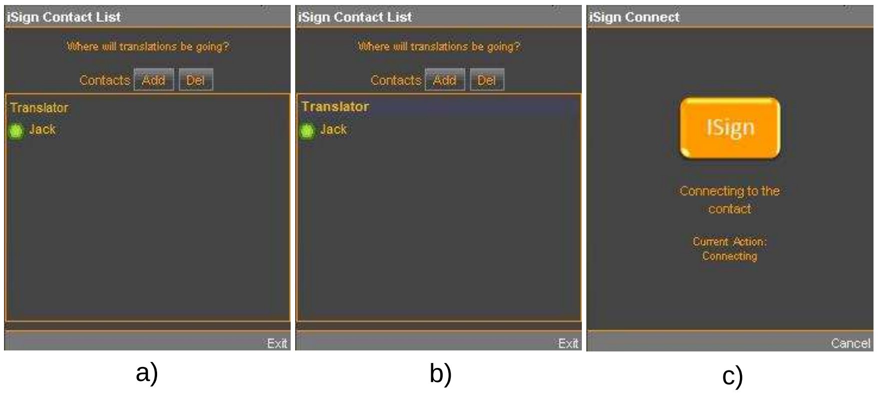


FIGURE 6.6: (a) The initial *Contacts* screen. (b) The *Contacts* screen with the 'Translator' entry selected. (c) Screen shown while sending information to the Translation Server and attempting to connect to the contact selected.

6.3.3.3 The *Mode-Select* Screen

The *Mode-Select* screen prompts the user to select the translation mode he/she would like to operate in. Deaf mode allows the user to capture and render SASL video as well as rendering a 3D Avatar signing SASL. Hearing mode allows the user to capture and render English audio. Figure 6.7 depicts the *Mode-Select* screen for Translator mode.

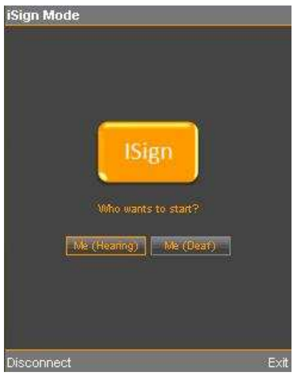


FIGURE 6.7: The *Mode-Select* screen for Translator mode.

6.3.3.4 The *Deaf Capture* and *Hearing Capture* Screens

Depending on the initial mode selected, either the *Deaf Capture* or *Hearing Capture* screens are displayed. The initial *Deaf Capture* and *Hearing Capture* screens are depicted in figures 6.8a and 6.9a respectively. It is possible to switch between these screens in the Translator mode by accessing the 'Navigate' menu available on both screens by pressing button 1.

On either screen, the user may press button 2 while the 'Record' button is highlighted. This activates a viewfinder window and audio capture window for the respective modes, depicted in figures 6.8b and 6.9b. The viewfinder allows the user to adjust the image before pressing button 2 to record. Once the stop button, depicted in figures 6.8c and 6.9c, has been pressed and the recording process is complete, the user may play, accept or cancel the recording using the buttons depicted in figures 6.8d and 6.9d. If the 'Accept' button is pressed, the respective *Capture* screens indicate that a recording is now loaded and ready to be sent to the Translation Server, as shown in figures 6.8e and 6.9e. The file can be sent by pressing the 'Send' button, which is now enabled. The screen depicted in figures 6.8f and 6.9f is displayed while the file is being sent.

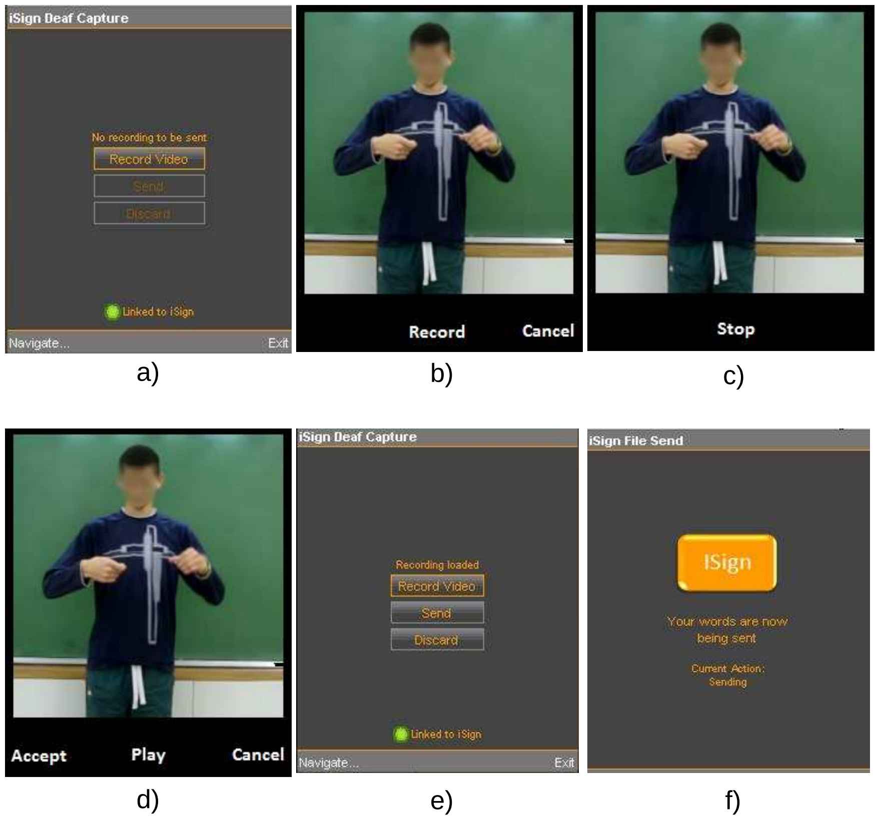
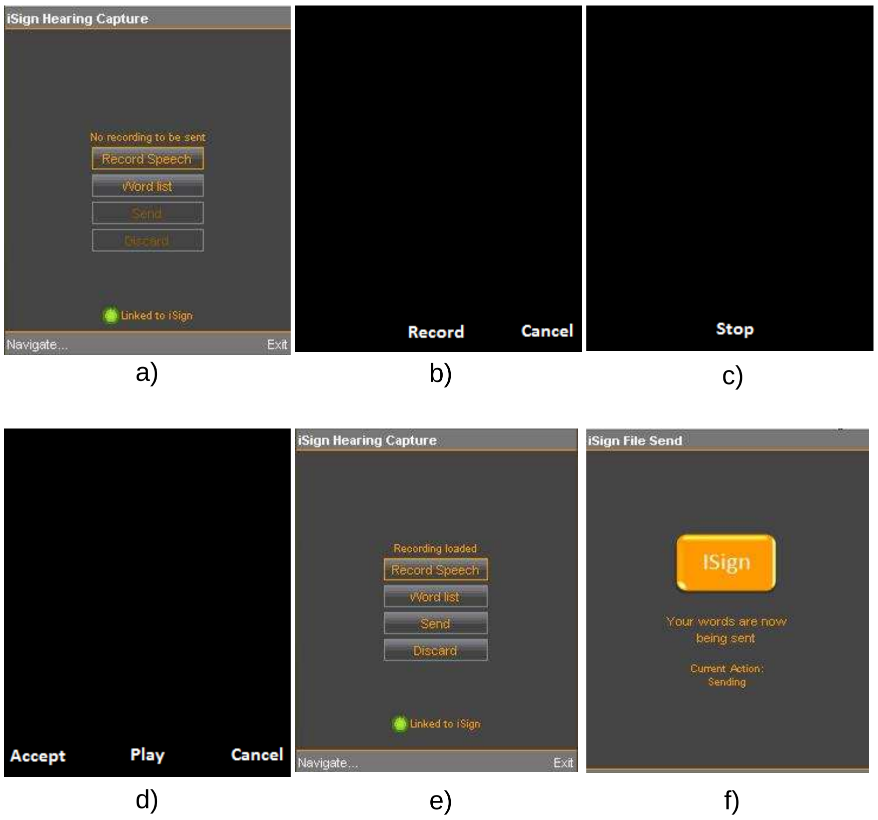


FIGURE 6.8: (a) The initial *Deaf Capture* screen. (b) The video viewfinder. (c) The video viewfinder in the process of recording. (d) The video viewfinder after recording is complete. (e) The *Deaf Capture* screen with a recording loaded. (f) The screen displayed while the file is being sent.



No recording to be sent

Recording loaded

Your words are now being sent

Current Action: Sending

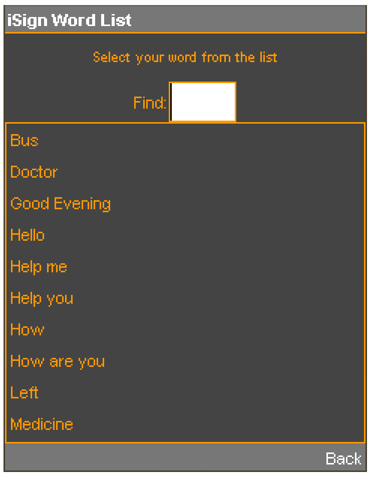
FIGURE 6.9: (a) The initial *Hearing Capture* screen. (b) The audio capture window. (c) The audio capture window in the process of recording. (d) The audio capture window after recording is complete. (e) The *Hearing Capture* screen with a recording loaded. (f) The screen displayed while the file is being sent.

6.3.3.5 The *Word List* Screen

As seen in Figure 6.9a, the *Hearing Capture* screen has a button 'Word List' that is not present in the *Deaf Capture* screen. Pressing this button invokes the *Word List* screen depicted in Figure 6.10. The *Word List* is a feature that allows hearing users to learn SASL by enabling them to perform a dictionary lookup of the English words provided. The user may select a word from the list and press button 2. The *Hearing Capture* screen indicates that a recording is loaded and ready to be sent as in Figure 6.9e. The word may be sent to the Translation Server as explained in the previous section to retrieve the SASL animation associated with that word. From interaction with SASL instructors and teachers, it became apparent that this feature could prove very useful, especially for hearing parents with deaf children.

6.3.3.6 Rendering Screens

Once a job has been sent and translated, the result may be retrieved and played back. The user is prompted whether or not to playback the message received as shown in



iSign Word List

Select your word from the list

Find:

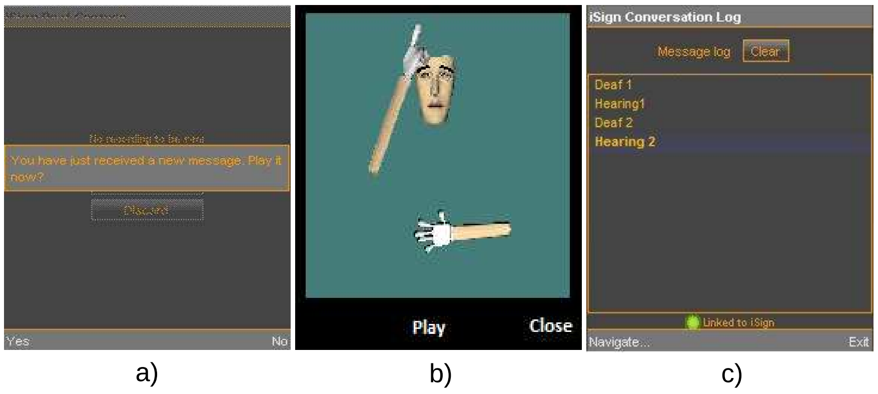
Bus
Doctor
Good Evening
Hello
Help me
Help you
How
How are you
Left
Medicine

Back

FIGURE 6.10: The *Word List* screen.

Figure 6.11a. We incorporated our Avatar *Phlank* into iSign for the rendering of SASL. *Phlank* is rendered and animated as a 3D model in the application. The playback of this Avatar is shown in Figure 6.11b. An audio playback screen is used to playback English audio that has been retrieved. This screen looks very similar to the screen in Figure 6.9d except that it has no 'Accept' button and has a 'Close' button in place of the 'Cancel' button in the figure.

Messages are stored in a log that can be accessed using the 'Navigate' menu, and is depicted in Figure 6.11c. The log can be used to play back captured messages and their corresponding translated messages. Messages are labelled according to the type of message sent and the order in which they were sent. Messages sent by Deaf capture are labelled as 'Deaf', and messages sent by Hearing capture are labelled 'Hearing'. Figure 6.11c depicts a log containing four messages.



You have just received a new message. Play it now?

iSign Conversation Log

Message log

Deaf 1
Hearing1
Deaf 2
Hearing 2

Linked to iSign

FIGURE 6.11: (a) The user is prompted whether or not to play the message retrieved. (b) The Avatar *Phlank* incorporated into iSign. (c) The message log depicting four messages.

6.3.4 The Interpretation Services

A phrase dictionary lookup was used in place of the interpretation systems since the SASL group has not yet produced systems in this area. This implementation can convert a set of 20 phrases between English and SASL.

The interpretation services in our prototype implementation consist of an extended version of Naidoo's Gesture Recognition system [88]. Naidoo adapted his system to conform to our prototype communication protocol. He built a timer into the system that polls the web service with the *getvidjob* and *getwavjob* signals explained in a previous section. The polling interval was set to 8 seconds. It handles one request at a time.

As mentioned before, his system carries out Gesture Recognition on SASL videos and produces a text file of the phrase recognized. His system was also extended to convert the text of the phrase into English audio using the Text To Wave ActiveX Client/Server DLL SDK v2.0 [100].

Naidoo's system was further extended to carry out speech recognition on English audio files using Microsoft Speech API (SAPI) 5.3 [81] to produce a text file of the phrase recognized. Our implementation places the conversion of English text to SASL Avatar on the phone using the *Phlank* Avatar. Therefore, the interpretation services for English to SASL interpretation in this prototype perform speech recognition and a dictionary lookup.

6.4 Summary

We have presented an overview of the conceptual SASL MT system framework of the SASL project as well as indicated those components that currently exist. We also presented our adaptation of this conceptual system into a mobile framework and explained our implementation of the mobile framework. This working prototype is usable and has been demonstrated at various technology events [1]. The mobile phone application meets the requirements specified in Chapter 3.

[1]This system won first place in the Software Design category in the Microsoft Imagine Cup South Africa 2008.

Chapter 7

Experimental Setup

In this chapter we describe the experiments we carried out. These experiments were aimed at determining whether or not synthetic sign language rendered on a mobile phone is intelligible to deaf people and investigating the cost factors involved in doing so. The two cost factors considered are the power and bandwidth usage.

For each experiment, we compared four methods of sign language visualization. The first method was the use of the low-detail Avatar *Phlank* rendered as a 3D model, depicted in Figure 7.1a. The second and third methods made use of the Avatar *Man*, depicted in Figures 7.1b and 7.1c. The difference between the second and third methods is that one does and one does not incorporate non-manual gestures. The final method was used as a comparative base and was the display of sign language videos of SASL speakers, depicted in Figure 7.1d. Henceforth, we refer to the methods as 'LowRes', 'Facial', 'NoFacial' and 'SLVid', respectively.

Each method was expected to have strengths and weaknesses in terms of intelligibility and cost factors. Our aim was to, both, determine the feasibility of each method, as well as compare the four methods.

The following sections describe each of the experiments carried out.

7.1 Sign Language Intelligibility

Sign language intelligibility was considered the most crucial feasibility factor in the use of mobile phones to visualize sign language.

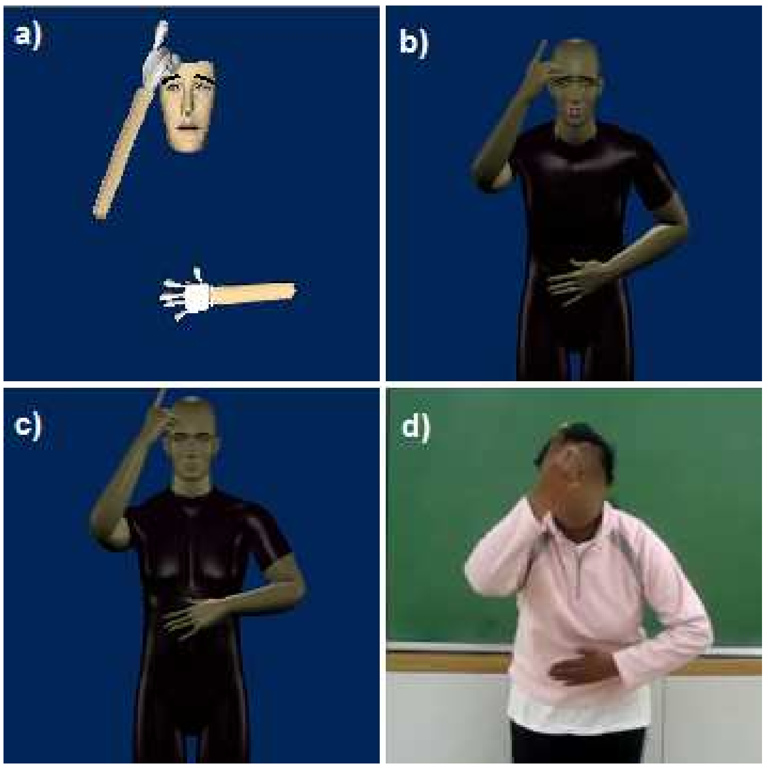


FIGURE 7.1: The SASL word 'Sick' as rendered by a) the low-detail Avatar *Phlank* (LowRes) b) the Avatar *Man* with non-manual gestures (Facial) c) the Avatar *Man* without non-manual gestures (NoFacial) and d) sign language video (SLVid).

7.1.1 Collection of SASL Videos and Exportation of Sign Language Files

In Chapter 5 we described the collection of our SASL videos from the Dominican School for the Deaf in Wynberg, Cape Town. We also described the method we used to animate and export the animations of the LowRes, Facial and NoFacial methods. We explained that the LowRes animations were exported as M3G model files that can be imported and rendered on the mobile phone as 3D models. We also explained that the Facial and NoFacial animations were too complex to be exported in the same manner and were exported as MPEG-4 Part 14 (MP4) format videos instead. The videos of the SLVid method were exported to the MPEG-4 Part 14 (MP4) format and imported into the Mobile Media API of Java ME and played as video files on the mobile phone.

7.1.2 Viewing Sequence

It was decided to use 16 test subjects. All 16 signs were shown once to each of the 16 test subjects yielding a total of 256 viewings. For each subject, the 16 viewings were divided into 4 groups, each of which was displayed using one of the 4 methods. This meant that each subject viewed 4 signs using each of the 4 methods.

The test subjects were also divided into 4 groups. Each group was shown a particular set of phrases with particular methods. Table 7.1 depicts the 4 viewing groups constructed such that each of the words in the word array was assigned to each of the methods in the corresponding method array, where words 1 to 16 are displayed in Table 5.7 in Chapter 5, and methods 1 to 4 correspond to LowRes, Facial, NoFacial and SLVid, respectively. This determined the word-method combinations for each group and ensured that each word-method would be viewed exactly 4 times by 4 different subjects. The viewing order for word-method combinations was then randomized for each subject to obtain 16 distinct viewing sequences. As an example, Table 7.2 shows the word-method viewing sequence of the first two subjects.

TABLE 7.1: Arrangement of method-word viewing groups.

Group	**Subjects**	**Arrangement**	
1	1, 5, 9, 13	Word:	{1,2,3,4,5,6,7,8,9,10,...,14,15,16}
		Method:	{1,2,3,4,1,2,3,4,1, 2 ,...,2 , 3 , 4}
2	2, 6, 10, 14	Word:	{1,2,3,4,5,6,7,8,9,10,...,14,15,16}
		Method:	{2,3,4,1,2,3,4,1,2, 3 ,...,3 , 4 , 1}
3	3, 7, 11, 15	Word:	{1,2,3,4,5,6,7,8,9,10,...,14,15,16}
		Method:	{3,4,1,2,3,4,1,2,3, 4 ,...,4 , 1 , 2}
4	4, 8, 12, 16	Word:	{1,2,3,4,5,6,7,8,9,10,...,14,15,16}
		Method:	{4,1,2,3,4,1,2,3,4, 1 ,...,1 , 2 , 3}

TABLE 7.2: Example viewing sequences of subjects 1 and 2.

		Viewing Number															
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Subject 1	**Word**	9	5	11	1	8	2	13	16	12	4	6	10	7	3	15	14
	Method	1	1	3	1	4	2	1	4	4	4	2	2	3	3	3	2
Subject 2	**Word**	3	4	2	9	7	16	13	12	6	5	8	14	10	1	11	15
	Method	2	3	1	4	2	3	4	3	1	4	3	1	1	4	2	2

7.1.3 Experimental Setup

In Chapter 2, we explained the experimentation carried out by Cox *et al.* of the ViSiCAST project to determine the intelligibility of the sign language rendered by their Avatar [29]. In [29], Cox *et al.* remark that variations in sign language dialect posed a serious challenge to the intelligibility of the sign language rendered by their Avatar. Taking note of this, we attempted to limit the effect of dialect variations by carrying out our experimentation in the same school from which we collected our signs.

Table 7.3 summarizes information pertaining to all 16 test subjects. The 16 test subjects consisted of 9 students, 3 teachers and 4 staff members. All 9 students were profoundly deaf and used SASL as their first language. 5 of them were females and 4 males. The students were in Grades that ranged from Grade 5 to Grade 9. The teachers were all hearing but had expert knowledge in SASL. 2 were female and 1 male. The staff members were all female. 3 were profoundly deaf and used SASL as their first language and 1 was hearing but had extensive experience and knowledge of SASL. As seen in Table 7.3, the subjects were tested in no particular order.

Table 7.3: Test subjects used in the intelligibility experiment.

Subject	**Gender**	**Type**	**Grade**	**Deaf**
1	M	Teacher	-	No
2	M	Student	9	Yes
3	M	Student	8	Yes
4	F	Student	5	Yes
5	F	Staff	-	No
6	F	Staff	-	Yes
7	F	Teacher	-	No
8	F	Student	7	Yes
9	F	Staff	-	Yes
10	F	Student	9	Yes
11	F	Teacher	-	No
12	M	Student	8	Yes
13	F	Staff	-	Yes
14	M	Student	6	Yes
15	F	Student	5	Yes
16	F	Student	5	Yes

Our experimental setup was an adaptation of the experimental setup of Cox *et al.* of the ViSiCAST project [29]. A teacher acted as an interpreter between the experimenter and the deaf subjects. A consent form was presented to each subject and the contents thereof were explained to the subject by the teacher. The form explained the purpose of the research as well as of the fact that participation was voluntary and it was possible to withdraw from the exercise at any time.

The teacher was requested to explain the experimental procedure to the subjects. Each subject was handed an answer sheet containing a table with 16 rows marked from 1 to 16. He/she was shown the viewing sequence explained in a previous section. Each of the words in the sequence was shown without context. For each viewing, the subject was instructed to write down the phrase they had seen in the corresponding row in the answer sheet provided, or to write “I dont know” in the that row. Students that struggled with writing were assisted by an interpreter.

The subject was allowed to replay the sign up to a maximum of 10 times, after which the phrase was marked as unknown. Those signs that had been indentified incorrectly were then re-presented to the subject and the subject was informed of the meaning that we had assumed. He/she was then asked whether this discrepancy was due to inappropriateness of the sign or whether it was due to unclearness of the sign on the mobile phone screen. Inappropriateness included the sign being of a varied dialect and errors in the animation such as incorrect hand shape or motion.

The results of the four methods were compared amongst each other to determine whether or not they were significantly different. Logistic regression was used to analyse the dichotomous outcome variable where the outcomes were correct and incorrect identification. Least Squares Means were used to make pair-wise comparisons among the four methods. The null hypothesis stated that there was no significant difference between the methods while the alternate hypothesis stated that there was, in fact, a significant difference between the methods. These results were used to draw conclusions.

7.2 Power Consumption

We aimed to obtain an indication of the power consumption of each method. Power consumption may be affected by many factors. It is dependent on the mobile phone used in the experiment, the firmware on the mobile phone, and many other factors. We did not, therefore, aim to obtain absolute values. Rather, we resolved to determine the relative increase in power consumption between the idle state of the phone and the state in which sign language was being rendered using each of the four methods.

The experimental design is depicted in Figure 7.2. The mobile phone charger was connected to a 12V power supply. An ammeter was used to measure the current drawn and a voltmeter was used to measure the voltage in the circuit. It was found that the mobile phone could not operate without the battery in place, in spite of the phone being connected to the power supply with the charger. This fact is assumed to affect the result. To compensate for this, the experiment was carried out with the battery was charged to 100%. We only wished to obtain an indication of the relative power consumption. This setup sufficed for our purposes. Also, the phone that we used was found to automatically turn its display off after 5s of receiving no user input. It does not provide a mechanism to override this behaviour. Experimentation showed that key-presses caused increases in the current drawn and directly affected the result. It was decided to begin all measurements 10s after the display turned off.

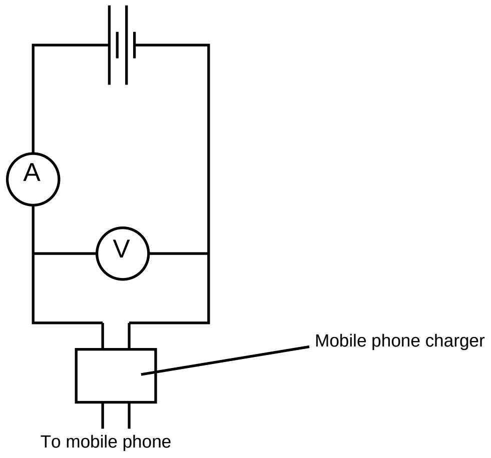


FIGURE 7.2: Experimental design used to determine power consumption.

The power consumption of the idle operation of the mobile phone were first measured. The mobile phone was left idle and the current and voltage were measured every second for 200s.

Thereafter, the power consumption of each SL visualization method was measured. Each phrase of each method was measured separately. The Java ME application was modified to play a selected phrase using a selected method repeatedly until this process was manually interrupted. The current and voltage were measured every second for 40s. This method was repeated for each phrase of each method. These values were then used to compute an average power consumption per method across all phrases.

The power consumption P in an electrical circuit of current I and voltage V is given by the mathematical expression in equation 7.1.

$$P = IV \tag{7.1}$$

Similar to the intelligibility experimentation, the results of the four methods were compared amongst each other to determine whether or not they were significantly different. Pair-wise comparisons were carried out between the methods by means of hypothesis testing using a Z-test with a 95% confidence interval. The null hypothesis stated that there was, in fact, a significant difference between the pair being compared, in terms of the current drawn, while the alternate hypothesis stated that this difference was not significant. These results were used to draw conclusions.

7.3 Bandwidth Consumption

The bandwidth consumption of each method is directly related to the file size of the animations. The files for SLVid, Facial and NoFacial were all video files in the format described in a previous chapter while those of LowRes were M3G files. We determined the size of the file for each phrase in our dictionary. This was used to compute an average. The averages of the four methods were then compared.

7.4 Summary

In this chapter, we have described the experiments we carried out in testing our research question. We mentioned the three requirements of a feasible mobile phone-based communication tool for the deaf, namely, feasibility in power and bandwidth usage and the ability to render intelligible Sign Language. We also described the experiments that we carried out to test for each requirement in detail.

Chapter 8

Results and Data Analysis

In this chapter, we present the results of the experiments explained in Chapter 7 and present an analysis of those results.

8.1 Sign Language Intelligibility

For the purpose of the analysis, all answers marked as "I don't know" were treated as incorrect answers. Therefore, answers were strictly either correct or incorrect. Table 8.1 summarizes the number of correctly identified phrases and their percentages of the total per test subject. Figure 8.1 is a graphical depiction of this data. Out of a total 256 viewings, 168 viewings, about 65% of the total, were identified correctly. The mean number of phrases recognized correctly was 10.5, with a standard deviation of 1.8 in this value. The number of correctly identified signs per person ranged between 13, about 81% of the total, and 6, about 38% of the total. This indicates diversity in test subjects.

Table 8.2 summarizes the number of viewings that were required before an attempt at recognition was made. 181 attempts at recognition, about 70% of the total, were made after only 1 viewing and an additional 58 attempts, about 23% of the total, were made after only 2 viewings. This suggests that the majority of recognition attempts – 93% – were made with conviction. Only 1 recognition attempt was made after more than 5 viewings. It was made after 6 viewings.

Table 8.3 summarizes the number of correctly identified phrases and their percentages of the total per method and this data is depicted graphically in Figure 8.2. Surprisingly, the sign language videos (SLVid) did not achieve 100% recognition. It fell short of this expected value and scored 81% recognition. The Facial method was observed to achieve a recognition rate that was comparable to that of SLVid (73%). The remaining two

Table 8.1: Correctly identified phrases per test subject.

Subject	**Number Correct (out of 16)**	**Percentage Correct (%)**
1	9	56
2	13	81
3	12	75
4	6	38
5	12	75
6	13	81
7	8	50
8	11	69
9	11	69
10	10	62
11	10	62
12	12	75
13	11	69
14	10	62
15	10	62
16	10	62
Total	**168**	**65**

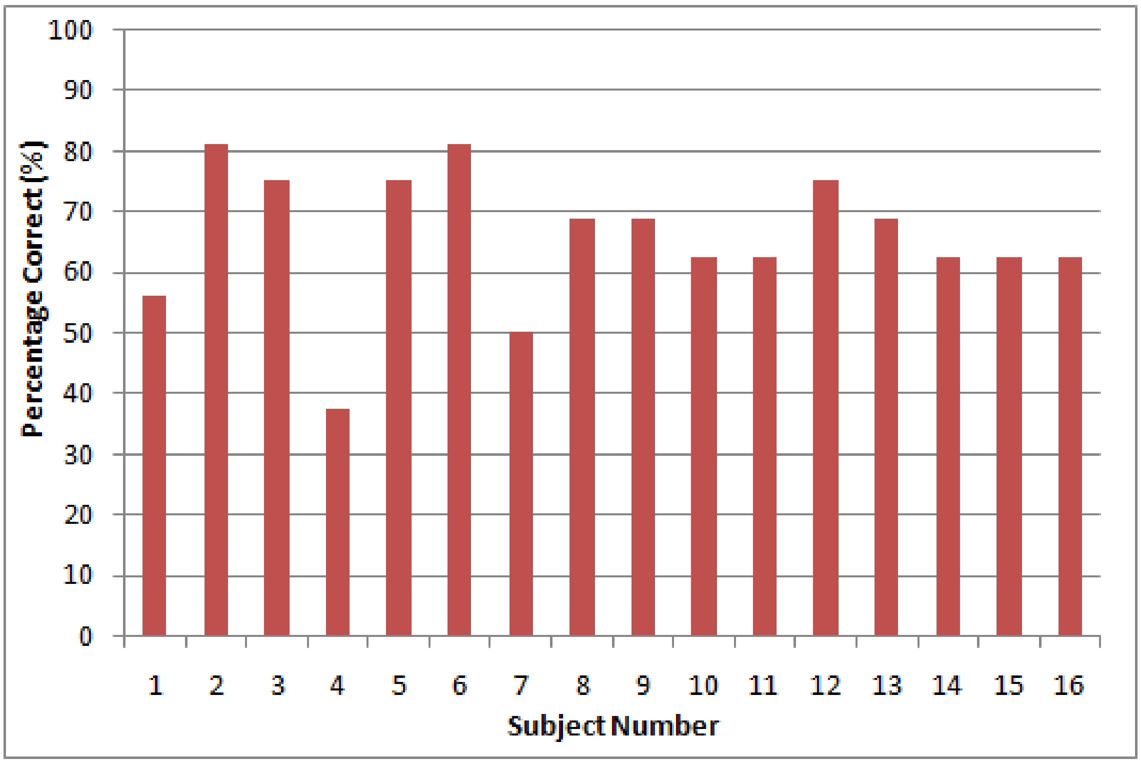


Figure 8.1: Percentage of correctly identified phrases per test subject.

methods, NoFacial and LowRes, achieved lower but similar recognition rates of about 54%.

We attempted to determine whether the differences between the recognition rates of

Table 8.2: Number of viewings required before an attempt at recognition was made.

		Number of Viewings						
		1	**2**	**3**	**4**	**5**	**6**	**Total**
Answer	**Correct**	146	17	3	1	0	1	**168**
	Incorrect	35	41	10	0	2	0	**88**
	Total	**181**	**58**	**13**	**1**	**2**	**1**	**256**

Table 8.3: Correctly identified viewings per method.

Method	**Number Correct (out of 64)**	**Percentage Correct (%)**
SLVid	52	81
Facial	47	73
NoFacial	34	53
LowRes	35	55
Total	**168**	**65**

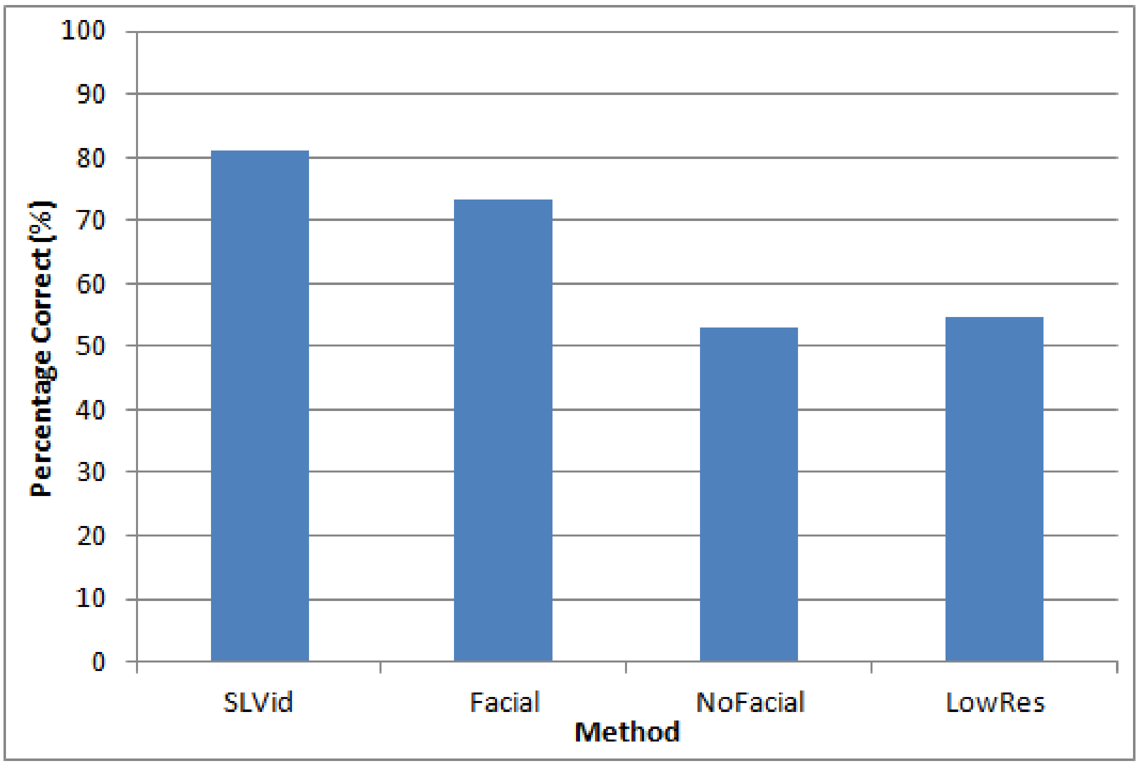


Figure 8.2: Percentage of correctly identified phrases per method.

the methods were statistically significant. We used logistic regression in analysing the dichotomous outcome variable where the outcomes were correct and incorrect identification. The fact that each subject provided repeated samples was accounted for. Least Squares Means were used to make pair-wise comparisons among the four methods. The null hypothesis stated that there was no significant difference between the methods while the alternate hypothesis stated that there was, in fact, a significant difference between the methods. Table 8.4 summarizes the results of carrying out pair-wise differences in

Least Squares Means for the methods with a 95% confidence interval.

Table 8.4: Differences of Least Squares Means for pair-wise comparisons between the four methods.

Method 1	**Method 2**	**Chi-Square**	**p-value**
Facial	**LowRes**	6.29	**0.0121**
Facial	**NoFacial**	8.23	**0.0041**
Facial	**SLVid**	1.76	**0.1852**
LowRes	**NoFacial**	0.11	**0.7381**
LowRes	**SLVid**	16.49	$<$ **0.0001**
NoFacial	**SLVid**	19.34	$<$ **0.0001**

It is observed that the Least Squares Comparison between recognition rates of the two method pairs SLVid-Facial and LowRes-NoFacial yield very high p-values of 0.1852 and 0.7381, respectively. Therefore, in line with our initial speculation, the difference in recognition rate between the two method pairs SLVid-Facial and LowRes-NoFacial is not statistically significant.

Table 8.5 summarizes the number of correctly identified viewings per phrase. This data is depicted graphically in Figure 8.3.

Table 8.5: Correctly identified viewings per phrase.

Phrase	**Number Correct (out of 16)**	**Percentage Correct (%)**
Help Me	16	100
South Africa	16	100
Toilet	16	100
Help You	15	94
Soccer	15	94
Good Evening	14	88
Hello	13	81
Water	12	75
Bus	11	69
Left	10	63
How	9	56
Right	8	50
Sick	6	38
Medicine	4	25
Doctor	2	13
Restaurant	1	6

It is observed that three phrases, ‘Help Me’, ‘South Africa’ and ‘Toilet’, achieved 100% recognition. There was a wide range in recognition over the 16 phrases but no phrase was completely unrecognizable. Only four phrases, ‘Sick’, ‘Medicine’, ‘Doctor’ and ‘Restaurant’, fell below 50% recognition. Figure 8.4 depicts the number of correct identifications for each method, for each of these four phrases. As an example, it can be seen in Figure

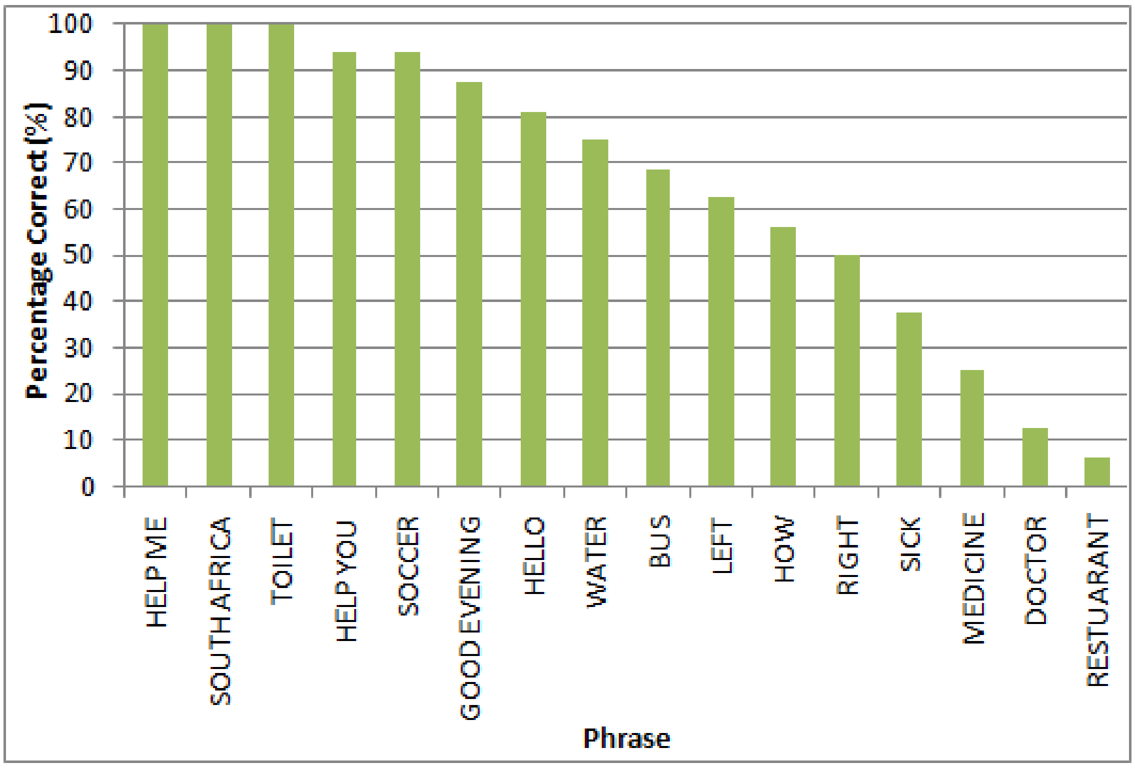


Figure 8.3: Percentage of correctly identified viewings per phrase.

8.4 that the word 'Sick' had a total of 6 correct identifications, 2 in the SLVid and Facial methods each, and 1 in the NoFacial and LowRes methods each. The most poorly performing phrase was 'Restaurant' with only one correct identification in the SLVid method. This phrase performed poorly across all methods and it is apparent that dialect variations in the school was the primary cause of the low recognition rate in this phrase.

While performing the experiment, it was observed that subjects regularly and repeatedly identified the phrase 'Sick' as 'Doctor' and the phrase 'Doctor' as 'Sick'. This was the biggest cause of recognition errors in these two words. This initially suggested that these signs may have been confused and mis-labelled at the time of collection. On the other hand, the word 'Sick' scored 6 correct identifications. We learned from the teachers that different SASL dialects use these two signs interchangeably and they are identified in context.

The word 'Medicine' scored 100% correct recognition using the SLVid method and 0% in all other methods. Comments from test subjects indicated that a detail in the shape and motion of the right hand in our animations changed the meaning of the sign from 'Medicine' to 'Cool drink'. This was therefore an error with the animation of the sign.

At first glance, it appeared that non-manual gestures played a major role in sign language intelligibility. The methods that included non-manual gestures (SLVid and Facial)

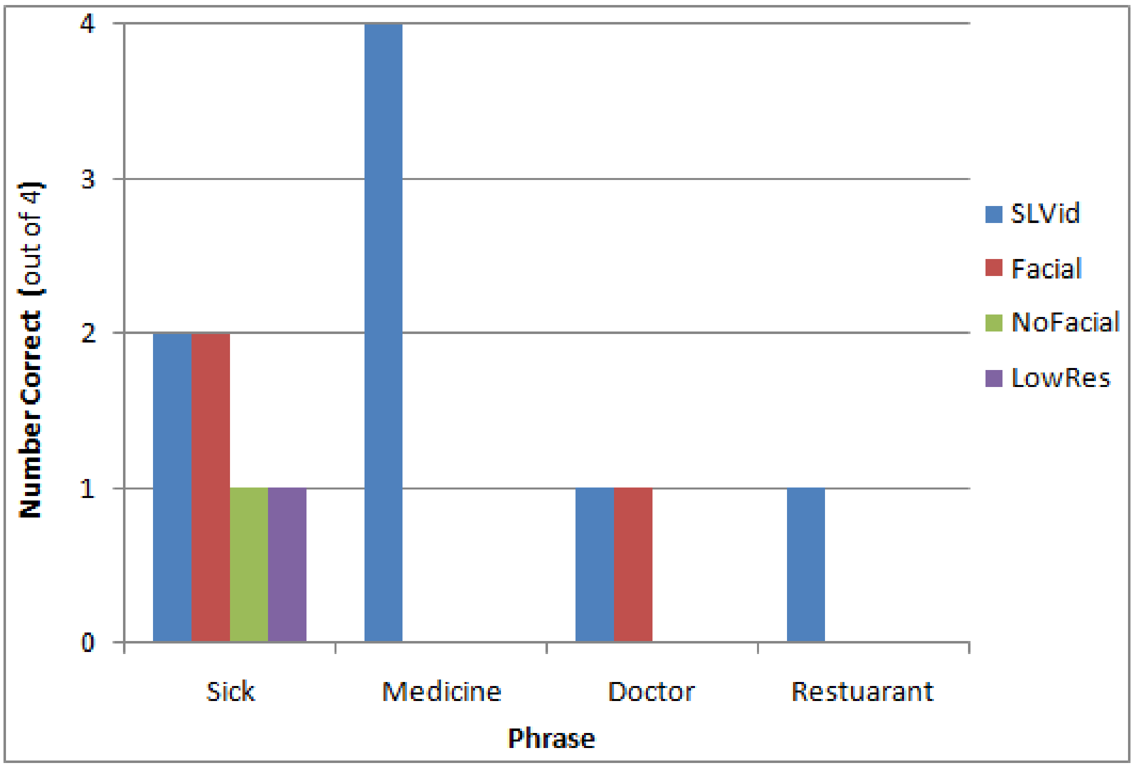


Figure 8.4: Number of correct identifications per method for the four phrases with recognition rates of less than 50%.

performed at least 20% better than those that did not (NoFacial and LowRes). However, closer observation of the data revealed that the latter methods fell short in a few specific phrases rather than in general. The most prominent of these phrases were 'How' and 'Water', and to some extent, 'Right'. Figure 8.5 depicts the graphs of correct identifications per method for each of these words. It is observed that, overall, the LowRes and NoFacial methods performed significantly worse than SLVid and Facial for these phrases. Across the three phrases, the SLVid and Facial methods scored a combined recognition rate of 92% whereas NoFacial and LowRes scored a combined recognition rate of 29%. This suggests that non-manual gestures may play a significant role only in certain phrases.

Even though the percentage of correctly identified phrases per subject, per method and overall are very encouraging, and suggest that we can in fact successfully display intelligible sign language on mobile phones using all four methods, we analyze the reasons for incorrect identifications. As mentioned in our experimental setup, subjects were prompted to decide whether incorrect identifications made were due to the sign being inappropriate or unclear. Figure 8.6 depicts the number of incorrect classifications that were attributed to each of these factors for each phrase.

It is very clear that the majority of incorrect identifications – 61 viewings, about 70% of

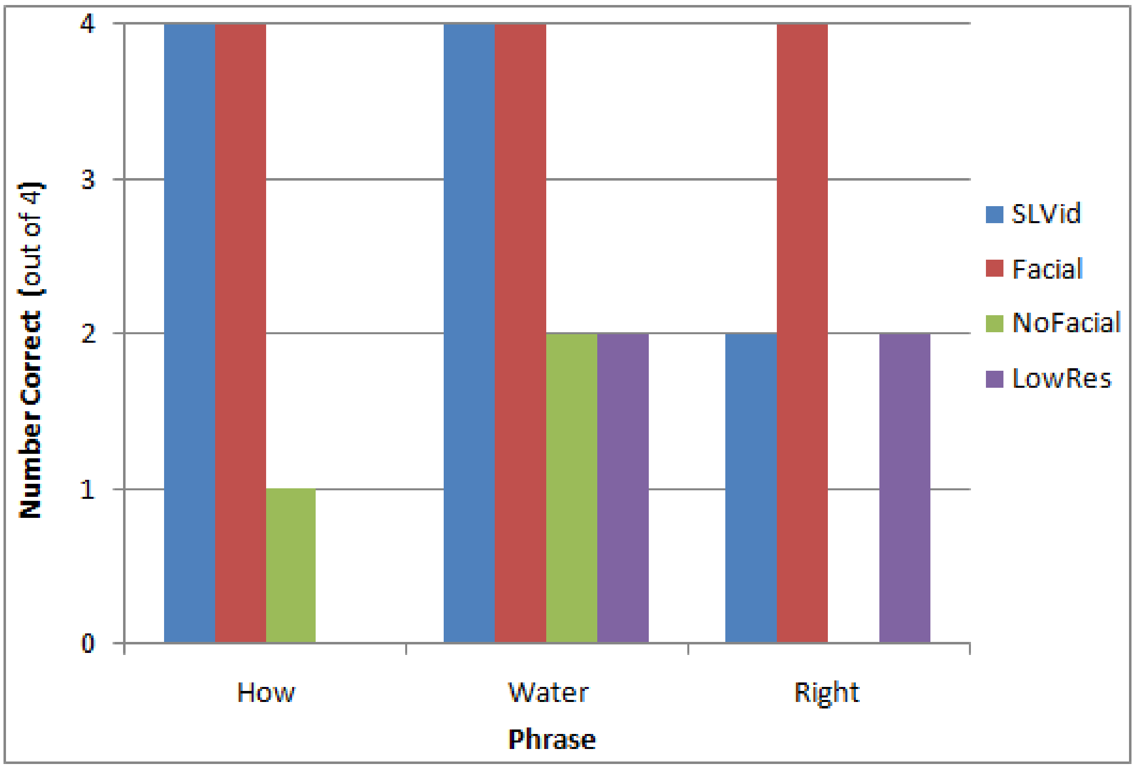


Figure 8.5: Number of correct identifications per method for the phrases in which SLVid and Facial performed better than NoFacial and LowRes.

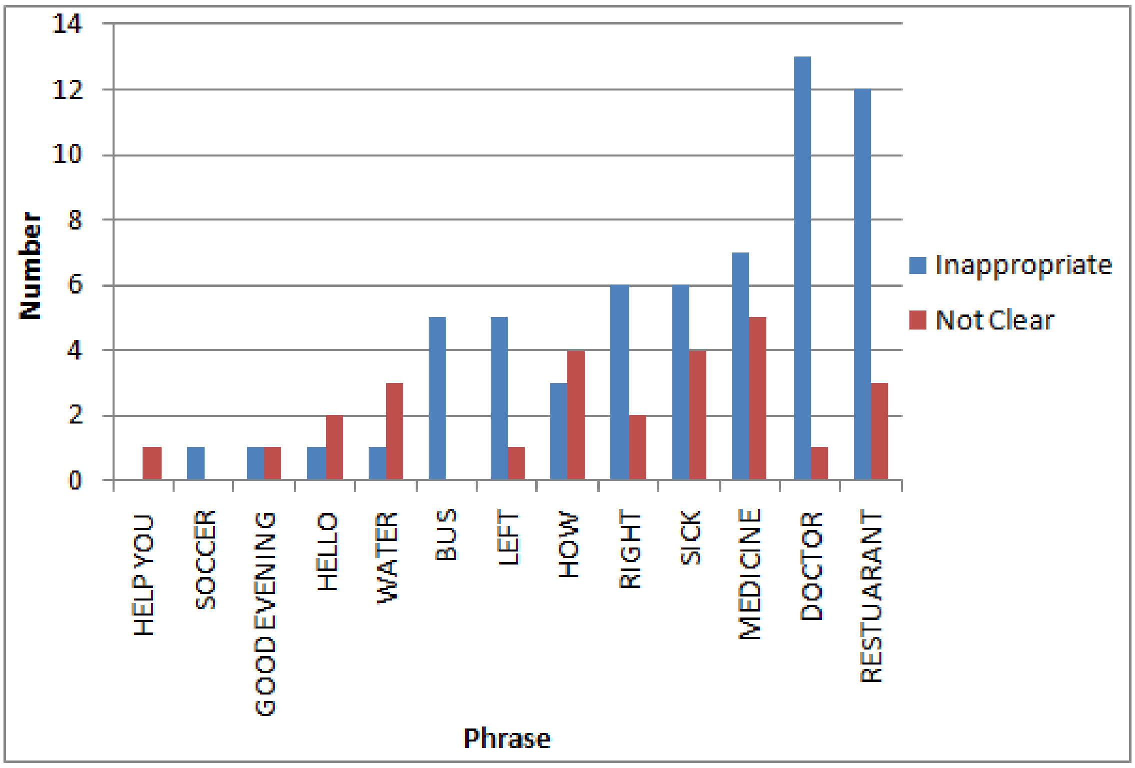


Figure 8.6: Frequency of reasons given for incorrect identification of phrases.

incorrect identifications – were due to inappropriateness of the sign, and only 27 viewings, about 30% of incorrect identifications, were due to unclearness of the sign on the mobile phone screen. Therefore, it has clearly been demonstrated that the size limitation on the mobile phone screen is not a significant drawback to displaying intelligible sign language on the mobile phone. Also, with the exception of the phrase 'Medicine' which registered a low recognition rate due to incorrect animation and phrases such as 'How' which suffered low recognition rates due to a lack of non-manual gestures, all remaining incorrect identifications were found to be caused by test subjects not being familiar with the dialect of phrases. This was in spite of our efforts to eliminate this factor, as has been explained. Comments such as "I don't use that sign", "I do that sign like this..." and "That sign is mostly used by this other group" were oft repeated. Cox *et al.* [29] also found this factor to be a significant challenge.

Subject 4, that had the lowest number of correct identifications, revealed that she had only recently joined the school and the dialect she used differed in many ways to the one used in the school. We also learned from multiple teachers that students had a sub-dialect of their own that had non-standard variations of certain signs.

We have stated our results as is. For investigative purposes, we wished to obtain an indication of the intelligibility of the four methods in the absence of factors that did not relate to clearness on the mobile phone screen. We eliminated all samples, correct and incorrect, of the four problematic words 'Sick', 'Doctor', 'Medicine' and 'Restaurant', all of which rated poorly due to factors that were irrelevant with respect to the experimental variable, as mentioned in detail. A total of $4 \times 16 = 64$ samples were removed reducing the total number of samples from 256 to 192. Table 8.6 summarizes the updated percentage of correctly indentified phrases after this removal.

TABLE 8.6: Updated number of correctly identified viewings per method after removal of problematic words.

Method	**Number Correct (out of 48)**	**Percentage Correct (%)**
SLVid	44	92
Facial	44	92
NoFacial	33	69
LowRes	34	71
Total	**155**	**81**

It is observed from Table 8.6 that all results increase significantly with the overall recognition rate increasing from 65% to 81%. The recognition rates for all methods is observed to have increased significantly. It is very interesting to note that, even after this removal, the trend in recognition rates of the methods remains consistent such that SLVid and Facial have the same recognition rate and NoFacial and LowRes have the same recognition rate, as previously proven.

8.2 Power Consumption

It was found that the voltage was constant to a high degree accuracy across all methods, including the idle state of the phone. The average Voltage was 10.04V with a 0% standard deviation in all significant figures of this result. This was not the case with the current. Since power is the product of voltage and current – as given in equation 7.1 – and the voltage was found to be constant, the power is directly proportional to the current. We therefore utilize the current alone to determine an increase in power consumption between the idle state and the four SL visualization methods.

The average amount of current drawn in the idle state was 10.7 mA with a standard deviation of 1.5mA. The average amount of current drawn $\bar{I}$ for each phrase of each method and the standard deviation in the mean s is summarized in Table 8.7.

Clearly, all three methods that rendered SL as video – SLVid, Facial and NoFacial – drew very comparable amounts of current. LowRes, on the other hand, drew a different and lower amount of current. This is further strengthened by the data in Table 8.8 which summarizes the average amount of current drawn $\bar{I}$ over all phrases for each method and the standard deviation in the mean s .

We sought to determine whether the current drawn by the three video-based methods were statistically equivalent, in accordance with observation. We also sought to determine whether the three video-based methods did in fact statistically differ from LowRes in this respect, in accordance with observation. We carried out pair-wise comparisons between the methods by means of hypothesis testing using a Z-test with a 95% confidence interval. The null hypothesis stated that there was, in fact, a significant difference between the pair being compared, in terms of the current drawn, while the alternate hypothesis stated that this difference was not significant. Table 8.9 summarizes the Z-test result for each pair-wise comparison.

As per our initial speculation, it is observed that the pairs Facial-NoFacial, Facial-SLVid and NoFacial-SLVid have very large p-values and support the alternate hypothesis that the difference in mean current drawn by all these methods is not significant – they are all equivalent. Therefore, for the remainder of this section, we treat them as one and refer to SLVid, Facial and NoFacial, collectively, as the video-based methods. The three pairs involving LowRes are observed to have very small p-values and support the null hypothesis that the difference between LowRes and the other three methods is, in fact, significant.

It should be noted that, while it was expected that the video-based methods would all have comparable power consumption rates and would differ from LowRes, which displays

Table 8.7: Average amount of current drawn $\bar{I}$ for each phrase of each method and the standard deviation in the mean s.

Phrase	**SLVid**		**Facial**		**NoFacial**		**LowRes**	
	$\bar{I}$ **(mA)**	s **(mA)**	$\bar{I}$ **(mA)**	s **(mA)**	$\bar{I}$ **(mA)**	s **(mA)**	$\bar{I}$ **(mA)**	s **(mA)**
Bus	32.8	3.6	32.0	2.9	32.3	3.3	23.0	3.1
Doctor	31.7	2.5	30.8	2.9	32.2	3.1	23.7	2.7
Good Evening	32.1	2.3	30.8	3.6	32.2	3.7	23.2	2.5
Hello	33.0	3.1	32.0	3.0	31.2	2.9	23.0	2.7
Help Me	31.7	2.5	31.4	4.4	31.1	3.1	23.4	2.8
Help You	31.3	3.0	31.4	2.9	31.6	5.1	24.1	2.9
How	31.6	2.7	32.4	3.6	32.2	3.1	24.1	2.9
Left	31.8	2.3	32.7	3.4	32.5	3.0	24.1	3.5
Medicine	31.1	2.4	31.4	2.4	32.0	3.3	23.9	2.8
Restaurant	31.9	3.2	32.5	3.5	31.0	3.0	23.0	2.4
Right	31.2	2.3	32.0	2.7	32.6	3.5	23.7	2.5
Sick	32.0	3.4	32.7	3.8	31.7	3.6	23.9	2.5
Soccer	31.6	2.7	31.0	2.7	32.2	3.1	23.7	2.5
South Africa	32.1	2.5	33.2	3.2	32.1	3.3	25.7	3.5
Toilet	31.9	2.7	31.7	2.9	32.1	3.0	25.0	3.7
Water	31.9	2.7	31.7	2.9	32.1	3.0	23.8	2.4

Table 8.8: Average amount of current drawn $\bar{I}$ by each method and the standard deviation in the mean s.

Method	$\bar{I}$ **(mA)**	s **(mA)**	**95% Confidence Interval**
SLVid	31.9	2.8	(31.6 , 32.1)
Facial	31.9	3.3	(31.6 , 32.1)
NoFacial	31.9	3.4	(31.7 , 32.2)
LowRes	24.0	3.0	(23.6 , 24.1)

Table 8.9: Z-test result for each pair-wise comparison.

Method 1	**Method 2**	**t**	**p-value**
Facial	**LowRes**	45.51	<0.005
Facial	**NoFacial**	-0.66	0.73
Facial	**SLVid**	-0.07	0.95
LowRes	**NoFacial**	-45.68	<0.005
LowRes	**SLVid**	-48.30	<0.005
NoFacial	**SLVid**	0.63	0.53

SL as a 3D model, it was against expectation to note that the latter method had a lower power consumption than the former methods. It was expected that displaying SL as a 3D model would consume more power than displaying it as video since it requires extra processing such as computing vertex locations and motions.

Table 8.10 summarizes the increase in current drawn (and, therefore, the power consumption) between the idle state of the phone and the SL visualization methods. This data is depicted graphically in Figure 8.7. For the sake of consistency with the final result of other sections, the graph of Figure 8.7 treats each of the video-based methods separately.

Table 8.10: Increase in current drawn $\bar{I}$, standard deviation s and the percentage increase in current drawn.

Method	**Increase in $\bar{I}$ (mA)**	**s (mA)**	**Percentage increase (%)**
SLVid	21.2	3.4	198
Facial	21.2	3.7	198
NoFacial	21.2	3.7	198
LowRes	13.3	3.3	124

It should be noted that these results are likely device-specific and possibly only apply to the mobile phone we used.

8.3 Bandwidth Consumption

Table 8.11 summarizes the file size for each phrase in each method which is directly proportional to the bandwidth usage. This data is depicted graphically in Figure 8.8.

The individual file sizes are not of particular significance since they may vary depending on the complexity and duration of the animation as seen in the data. The comparison in file size between the methods for each phrase is of relevance and a trend is observed. It can clearly be seen that the file size for SLVid animations significantly exceed that of all other methods for all phrases. It can also be seen that Facial and NoFacial animation

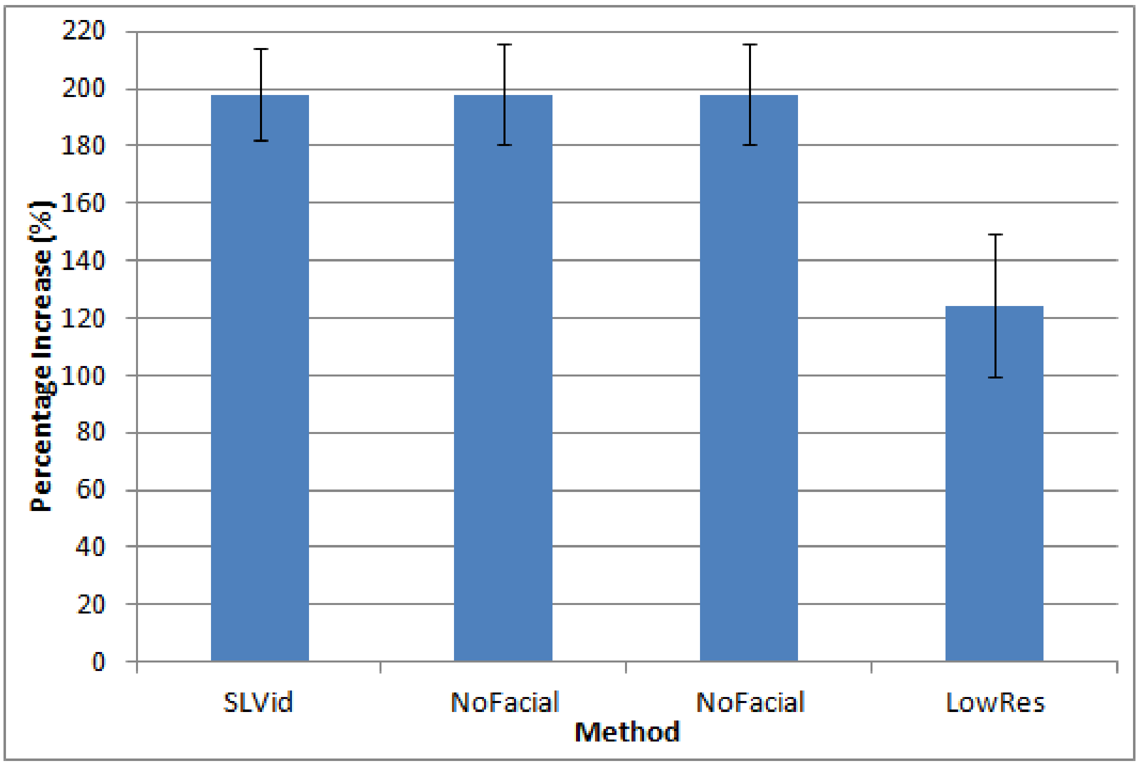


FIGURE 8.7: Percentage increase in power consumption by each method.

TABLE 8.11: File size for each phrase of each SL visualization method.

Phrase	File Size (Bytes) of method			
	SLVid	**Facial**	**NoFacial**	**LowRes**
Bus	358 845	127 845	126 742	88 786
Doctor	331 137	120 910	113 974	89 938
Good Evening	313 987	167 314	166 394	93 178
Hello	317 585	94 179	89 042	87 994
Help Me	265 067	147 690	146 199	91 234
Help You	275 540	153 751	152 552	91 306
How	310 837	160 820	136 081	95 122
Left	272 507	107 780	96 470	87 994
Medicine	360 854	185 147	154 501	91 234
Restaurant	283 002	138 142	129 401	91 234
Right	231 622	105 852	94 252	87 994
Sick	324 624	162 552	134 957	91 234
Soccer	251 791	129 235	128 085	91 234
South Africa	300 192	142 247	140 441	91 774
Toilet	222 723	141 216	140 514	91 234
Water	247 757	111 376	104 619	89 938
Total	**4 668 070**	**2 196 056**	**2 048 935**	**1 451 428**

files have comparable file sizes but animation files for Facial slightly exceed those of NoFacial for all phrases. LowRes animation files are seen to have the smallest file size for all phrases. Table 8.12 summarizes the average file size of each method and the

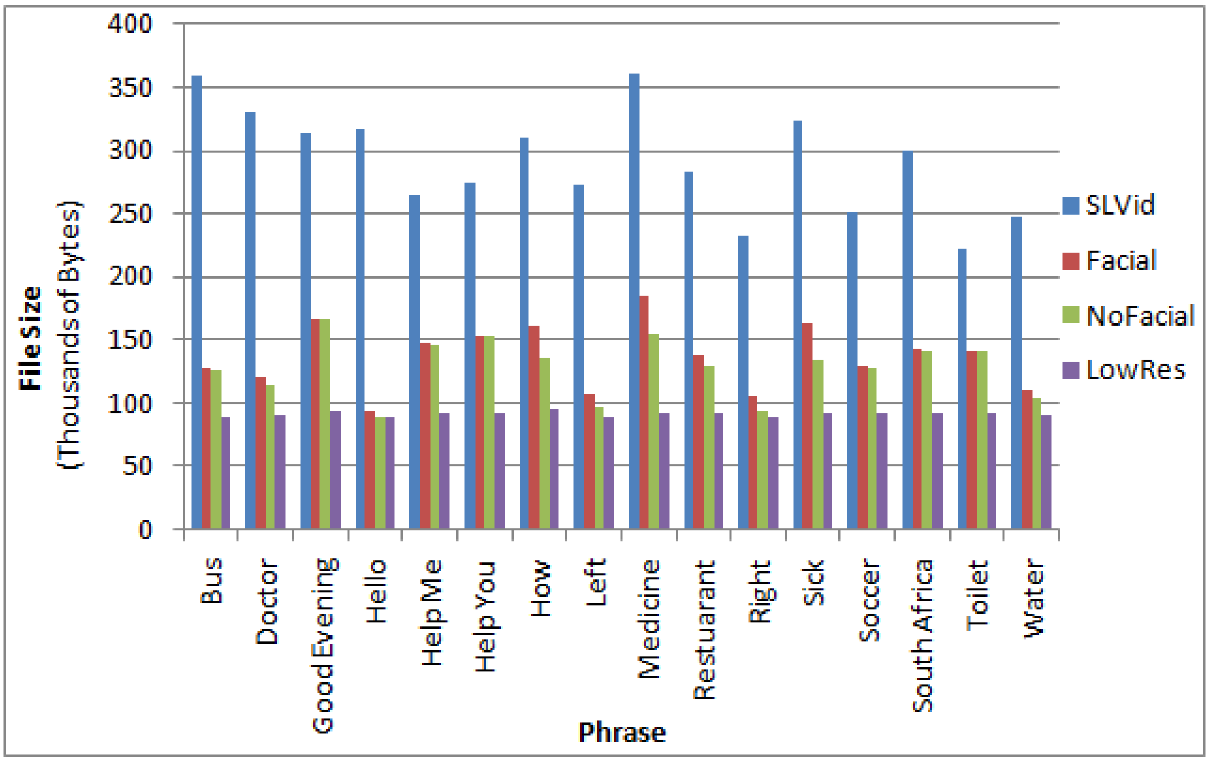


FIGURE 8.8: File size of each phrase for each SL visualization method.

percentage standard deviation from the mean, and this data is depicted graphically in Figure 8.9.

TABLE 8.12: Average file size of each method and the percentage standard deviation from the mean.

Method	Average File Size (Bytes)	Percentage Std. Dev. (%)
SLVid	291 754	14.5
Facial	137 254	18.4
NoFacial	128 389	17.9
LowRes	90 714	2.1

It is observed that SLVid, Facial and NoFacial have a large variation in file size of about 15%. This is not the case with LowRes which has a very low variation in file size of only 2% across all phrases. The large variation in file size for the methods that make use of video is attributed to the fact that as the complexity in the phrase increases, so does its duration and the number of frames in the video file. On the other hand, increases in complexity and duration of M3G animations are accounted for by appending only animation data of particular vertices that are involved in the added complexity to the original file.

It is also observed that Facial and NoFacial have comparable file sizes. Both, the file size and the variation in file size are observed to be similar. In this respect, it is observed

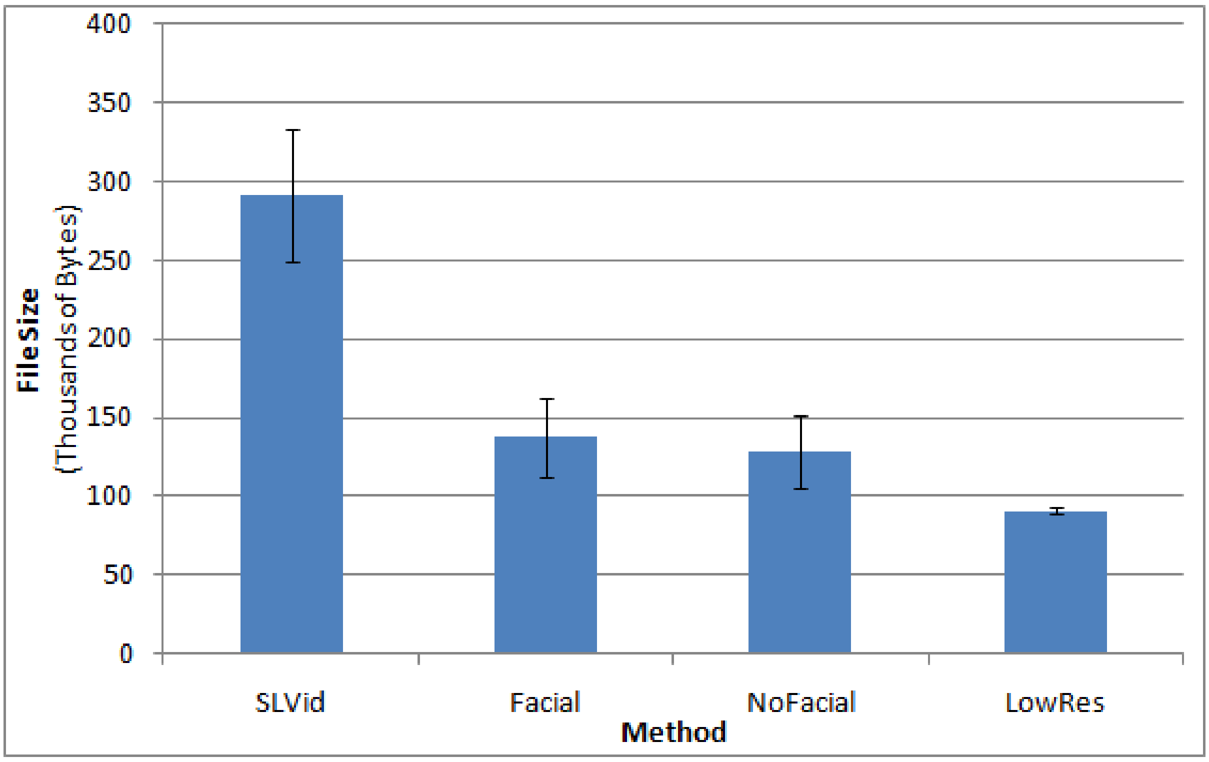


FIGURE 8.9: Average file size of each method.

that the inclusion of non-manual gestures does not have a significant impact on file size.

8.4 Summary of Results and Discussion

Table 8.13 summarizes the performance of each SL visualization method investigated in terms of the factors considered in this chapter.

LowRes provides a high intelligibility rate, although not as high as any other method, but has the lowest power and bandwidth consumption and the lowest variation in file size across phrases. Facial and NoFacial have comparable bandwidth and power consumption and variation in file size, but Facial has a higher intelligibility rate than NoFacial. The performance of these two methods generally falls in between LowRes and SLVid. We conclude that non-manual gestures do affect the intelligibility of the method but only for certain phrases. SLVid, while providing the highest intelligibility rate, also has the highest bandwidth consumption, as well as a high variation in file size and high power consumption as compared to other methods.

We have shown that all four methods can be used effectively in SL visualization on mobile phones with cost advantages and disadvantages.

Table 8.13: Summary of the performance of each SL visualization method in terms of the factors considered.

	SLVid	**Facial**	**NoFacial**	**LowRes**	**Average**
Intelligibility (%) (reduced external factors)	92	92	69	71	81
Intelligibility (%) (with external factors)	81	73	53	55	65
Power Consumption Increase (%)	198	198	198	124	180
Bandwidth Consumption (Bytes)	291 754	137 254	128 389	90 714	162 028
Variation in File Size (%)	14.5	18.4	17.9	2.1	13.2

8.5 Summary

We have presented the results of our experiments. Our analyses have indicated that it is, in fact, possible to use all four methods of SL visualization to effectively display SL on mobile phones.

Chapter 9

Conclusion and Directions for Future Research

This thesis has made several important contributions to the field of sign language visualization on mobile phones as well as Machine Translation between spoken and sign languages. We evaluated the state of APIs in the popular mobile development language Java ME and showed that many mobile phones provide the APIs required to use them as service-delivery devices for sign language Machine Translation systems.

We also showed, for the first time, that it is possible to display intelligible South African Sign Language (SASL) on mobile phones using a variety of methods. We further evaluated the power consumption and bandwidth usage of displaying SASL on the mobile phone. These results are an invaluable asset to any project that focuses on developing mobile phone-based sign language visualization methods.

Last but not least, we developed the first prototype system that provides automated translation from spoken to sign language **and** sign to spoken language, with the service delivered to users on a mobile phone. In so doing, we demonstratively showed that the use of mobile phones as service-delivery devices in such a translation system is feasible.

We provide two sets of directions for future research.

9.1 Feasibility Experimentation

In this thesis we made use of the Sony Ericsson C905. This mobile phone has an average size screen. Trends in mobile phone technology are observed to be slanting heavily towards large displays such as all modern iPhones, the Sony Ericsson Satio and

XPERIA, the HTC Magic and Desire and the Samsung S8000 Jet. We recommend that the intelligibility testing be extended to different display sizes.

Similarly, we recommend that the power consumption associated with the four sign language visualization methods be investigated on a variety of mobile phones as a comparison with our results.

9.2 The iSign Mobile Framework

The iSign Mobile Framework has been developed with scalability in mind. All components of the framework may be subject to development. We recommend that switching to a standard protocol such as the Session Initiation Protocol (SIP) be investigated. This may make it possible to link the iSign Framework to other communication systems that use SIP.

The interpretation services in the framework may also be improved. Many systems that the SASL Group has produced have not yet been incorporated into the framework. Examples are the hand-shape recognition system of Segers [106], the facial expression recognition and classification system of Whitehill [144], the gesture recognition system of Rajah [99] and Van Wyk's sign language visualization system *Man* [146]. Incorporating these systems into the framework will provide a more robust translation service.

Also, many requirements were taken into consideration when developing the iSign mobile application. Simplicity and ease of use were two very important considerations taken into account but were not a focus of the research. We suggest that usability testing be carried out on the iSign mobile phone application to cater for the special needs of the deaf, if required.

9.3 Concluding Remarks

This research has served as an extremely educational experience for this researcher. Our iSign Mobile Framework is a significant milestone for the SASL Group. We hope that this system, as well as our feasibility study, serves as a firm foundation for research and learning of many other researchers in the same group and worldwide.

Appendix A

The "TestModel" Application

Listings A.1 and A.2 are source code listings for the two files TestModel.java and TestModelCanvas.java of the TestModel application. The TestModel class is the MIDlet.

LISTING A.1: TestModel.java

```
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Display;
import javax.microedition.midlet.MIDlet;
import javax.microedition.lcdui.Command;
import javax.microedition.midlet.MIDletStateChangeException;

public class TestModel extends MIDlet implements CommandListener
{

         private static MIDlet self = null;
         private TestModelCanvas thecanvas = null;
         private Display display = null;


         protected void startApp() throws MIDletStateChangeException
         {
                  display = Display.getDisplay(this);
                  thecanvas = new TestModelCanvas();

                  thecanvas.addCommand(new Command("Quit", Command.EXIT, 1));

                  thecanvas.setCommandListener(this);

                  thecanvas.start();
                  display.setCurrent(thecanvas);

                  self = this;
         }

    public void commandAction(Command c, Displayable d) {
        if(c.getCommandType() == Command.EXIT)
```

```
            notifyDestroyed();
    }

        protected void pauseApp()
        {

        }

        public static void die()
        {
                self.notifyDestroyed();
        }

        protected void destroyApp(boolean unconditional)
                throws MIDletStateChangeException
        {
                notifyDestroyed();
        }
}
```

Listing A.2: TestModelCanvas.java

```
import javax.microedition.m3g.Graphics3D;
import javax.microedition.m3g.Camera;
import javax.microedition.m3g.Transform;
import javax.microedition.m3g.Light;
import javax.microedition.m3g.World;
import javax.microedition.m3g.Loader;
import javax.microedition.m3g.Object3D;
import javax.microedition.lcdui.Graphics;
import javax.microedition.lcdui.game.GameCanvas;


public class TestModelCanvas extends GameCanvas implements Runnable {

        World world = null;
    Graphics3D g3d = null;


    public TestModelCanvas()
    {
        super(true);

        setFullScreenMode(true);

        loadWorld();
    }


    private void loadWorld()
    {

        try
        {
            Object3D[] tmpworld = Loader.load("/model.m3g");
            for(int i = 0; i < tmpworld.length; i++)
            {
                if(tmpworld[i] instanceof World)
                {
                    world = (World)tmpworld[i];
                    break;
                }
            }

            tmpworld = null;
        }
        catch(Exception e)
        {
            System.out.println("Loading error!");
            printe(e);
        }
    }

    private void draw(Graphics g)
    {
        try
        {

         g3d = Graphics3D.getInstance();

         g3d.bindTarget(g, true, Graphics3D.ANTIALIAS
                        | Graphics3D.TRUE_COLOR | Graphics3D.DITHER);

         g3d.render(world);
        }
        catch(Exception e)
        {
            printe(e);
        }
        finally
        {
            g3d.releaseTarget();
        }
    }

    public void start() {
        Thread trd = new Thread(this);
        trd.start();
    }

    public void run() {
        while(true) {
            try {

                draw(getGraphics());
                flushGraphics();

                try{ Thread.sleep(30); } catch(Exception e) {}
            }
            catch(Exception e) {
                printe(e);
            }
        }

    }


    private void printe(Exception e) {
        System.out.println(e.getMessage());
    }
}
```

Appendix B

The iSign Communication Protocol

B.1 Signals

Tables B.1, B.2, B.3, B.4 and B.5 summarize the parameters, return values and description of each signal of the communication protocol. The signals have been sub-divided according to the task to which they apply.

TABLE B.1: Signal pertaining to *Login* on the phone.

Signal	Parameters	Return Values	Description
login	–cellno	–sessionid ⟨or⟩ –“Error”	Sends the pre-stored phone number and pin. If login is successful, a session id is returned. If not the phrase “Error” is returned.

TABLE B.2: Signals pertaining to the *Connect* on the phone.

Signal	Parameters	Return Values	Description
contacts	–sessionid	–[List of contacts]	Retrieves a list of phones registered on the framework that are currently logged in and available.
connect	–sessionid –otherno	–“Waiting” ⟨or⟩ –“Not available”	Attempts to connect to a phone. Return values correspond to the target phone being unavailable and waiting for it to respond to the connect request.
setmode	–sessionid –mode	–Null ⟨or⟩ –“Error”	Sets the current capture and render mode to either deaf (SASL) or hearing (English) mode. Returns “Error” if the connection was interrupted.

Table B.3: Signals pertaining to the *Job Upload* on the phone.

Signal	**Parameters**	**Return Values**	**Description**
vidin	–sessionid –otherno –[Video File] –fileno	–"OK" ⟨or⟩ –"Error"	Sends a SASL video message to be processed and a response sent to the target phone. Return values correspond to success and failure of the upload respectively.
wavin	–sessionid –otherno –[Audio File] –fileno	–"OK" ⟨or⟩ –"Error"	Sends an English audio message to be processed and a response sent to the target phone. Return values correspond to success and failure of the upload respectively.

Table B.4: Signals pertaining to the *Job Processing* on the server.

Signal	**Parameters**	**Return Values**	**Description**
getvidjob	–None	–"None" ⟨or⟩ –[directory, destmode, fileno, clientno]	Queries for any unprocessed video jobs. Returns "None" if there are no such jobs or information on the file to be retrieved.
getwavjob	–None	–"None" ⟨or⟩ –[directory, destmode, fileno, clientno]	Queries for any unprocessed audio jobs. Returns "None" if there are no such jobs or information on the file to be retrieved.
vidjobdone	–fileno –clientno	–Null	Notification of successful completion of SASL to English interpretation processing.
wavjobdone	–fileno –clientno	–Null	Notification of successful completion of English to SASL interpretation processing.
vidjobfailed	–fileno –clientno	–Null	Notification of unsuccessful completion of SASL to English interpretation processing.
wavjobfailed	–fileno –clientno	–Null	Notification of unsuccessful completion of English to SASL interpretation processing.

B.2 Database Tables

Tables B.6, B.7, B.8 and B.9 are data dictionaries for the tables in the MySQL database.

Table B.5: Signal pertaining to the *Job Retrieval* on the phone.

Signal	Parameters	Return Values	Description
querymsg	–sessionid	‘–‘None” ⟨or⟩ –[URL of interpreted file to be retrieved]	Queries for any interpreted messages directed at this phone. Returns “None” or a URL to the file to be retrieved.

Table B.6: Data dictionary of the USER table.

Field	Type	Description
no	int(10)	Primary key: Unique integer assigned to each phone registered on the service.
cellno	text	Cell phone number of the mobile phone.
pin	text	Pin number of the mobile phone, used when logging in.
nick	varchar(20)	Nickname specified by the mobile phone user, to be displayed to other contacts.
session	text	Uniquely generated session id generated according to the time of login by the PHP web-service.
accesstime	datetime	Date and time of login.
connect	int(10)	Field used to connect to other contacts or be connected to. Values are: 0 (available for connection), -1 (not availble, connecting to another contact), [any integer higher than 0] (not available - received connection request from phone [integer greater than 0]).
isignmode	enum(‘0’, ‘1’, ‘2’)	Stores the capture and render mode of the phone. Values are: 0 (Hearing mode), 1 (Deaf mode), 2 (Translator mode - both).

Table B.7: Data dictionary of the CONTACTS table.

Field	Type	Description
userno	int(10)	Foreign key of field ‘No’ in table User.
contactno	int(10)	Foreign key of field ‘No’ in table User.
active	enum(‘0’, ‘1’)	Specifies whether or not the invite sent by the contact specified by field ‘contactno’ has been accepted by the contact specified by field ‘userno’. It may also be used to block this contact after the invite has been accepted.

Table B.8: Data dictionary of the VIDEOIN table.

Field	**Type**	**Description**
no	int(10)	Foreign key of field 'No' in table User. Specifies the phone from which the unprocessed video job came.
destno	int(10)	Foreign key of field 'No' in table User. Specfies the phone to which the processed video job should be delivered.
fileno	int(10)	Integer uniquely identifying each video job sourcing from a phone. Incremented on the phone.
status	tinyint(3)	Specifies the status of the video job in the translation pipeline. Values are: 0 (unprocessed), 1 (under process), 2 (completed successfully), 3 (completed unsuccessfully).

Table B.9: Data dictionary of the WAVEIN table.

Field	**Type**	**Description**
no	int(10)	Foreign key of field 'No' in table User. Specifies the phone from which the unprocessed audio job came.
destno	int(10)	Foreign key of field 'No' in table User. Specfies the phone to which the processed audio job should be delivered.
fileno	int(10)	Integer uniquely identifying each audio job sourcing from a phone. Incremented on the phone.
status	tinyint(3)	Specifies the status of the audio job in the translation pipeline. Values are: 0 (unprocessed), 1 (under process), 2 (completed successfully), 3 (completed unsuccessfully).

Bibliography

[1] ISO/IEC 14496–2:2004, Information technology—Coding of audio-visual objects—Part 2: Visual.

[2] ISO/IEC 19775–1:2008, Information technology—Computer graphics and image processing—Extensible 3D (X3D). [Online] Available at http://www.web3d.org/x3d/specifications/ISO-IEC-19775-1.2-X3D-AbstractSpecification/index.html, (Accessed: January 2010).

[3] ISO/IEC FCD 19774:200x, Information technology—Computer graphics and image processing—Humanoid animation (H-Anim). [Online] Available at http://h-anim.org/Specifications/H-Anim200x/ISO_IEC_FCD_19774/, (Accessed: January 2010).

[4] ISO/IEC FDIS 14772–1:1997, Information technology—Computer graphics and image processing—The Virtual Reality Modeling Language (VRML)—Part 1: Functional specification and UTF-8 encoding. [Online] Available at http://www.web3d.org/x3d/specifications/vrml/ISO-IEC-14772-VRML97/, (Accessed: January 2010).

[5] L. Aarnio. Small-scale Java virtual machines. Citeseer. doi: 10.1.1.25.1651.

[6] A.K. Adesemowo and W.D. Tucker. Affective gesture feedback instant messaging on handheld. In *Proceedings of the 5th IEE International Conference on 3G Mobile Communication Technologies*, pages 499–503, 2004.

[7] O. Al-Jarrah and A. Halawani. Recognition of gestures in arabic sign language using neuro-fuzzy systems. *Artificial Intelligence*, 133(1–2):117–138, 2001.

[8] K.B. Atkinson. Close range photogrammetry and machine vision. Whittles, 1996.

[9] AT&T. AT&T video relay service official website. [online] available at http://www.attvrs.com/, (Accessed: April 2009).

[10] Australian Communication Exchange. Australian Communication Exchange's National Relay Service official website. [Online] Available at http://www.relayservice.com.au/, (Accessed: April 2009).

[11] Autodesk. Autodesk Maya. [Online] Available at http://usa.autodesk.com/sadsk/servlet/index?id=7635018&siteID=123112, (Accessed: November 2008).

[12] I. Baran and J. Popović. Automatic rigging and animation of 3D characters. In *Proceedings of the 2007 ACM SIGGRAPH International Conference*, page 72. ACM, 2007.

[13] M. Bastioni. New mesh model in MakeHuman 0.8. 2005. [Online] Available at http://www.dedalo-3d.com/docs/2005-12-01-new-model.pdf, (Accessed: June 2009).

[14] M. Bastioni, S. Re, and S. Misra. Ideas and methods for modeling 3D human figures. *ACM Bangalore*, 2008.

[15] F. Bernardini and H. Rushmeier. The 3D model acquisition pipeline. In *Computer Graphics Forum*, volume 21, pages 149–172. John Wiley & Sons, 2002.

[16] Blender Foundation. Blender official website. [Online] Available at http://www.blender.org/, (Accessed: January 2010).

[17] Blender Foundation. Blender version 2.49 official website. [Online] Available at http://www.blender.org/development/release-logs/blender-249, (Accessed: January 2010).

[18] W. Böhler and A. Marbs. 3D scanning instruments. *Proceedings of the CIPA WG*, 6:1–2, 2002.

[19] F.G. Bowe. Deaf and hard of hearing Americans' instant messaging and e-mail use: A national survey. *American Annals of the Deaf*, 147(4):6–10, 2002.

[20] H. Brashear, V. Henderson, K.H. Park, H. Hamilton, S. Lee, and T. Starner. American sign language recognition in game development for deaf children. In *Proceedings of the 8th International ACM SIGACCESS Conference on Computers and Accessibility*, page 86. ACM, 2006.

[21] F. Buttussi, L. Chittaro, and D. Nadalutti. H-animator: a visual tool for modeling, reuse and sharing of X3D humanoid animations. In *Proceedings of the 11th International Conference on 3D Web Technology*, page 117. ACM, 2006.

[22] E. Catmull and J. Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer-Aided Design*, 10(6):350–355, 1978.

[23] A. Cavender, R.E. Ladner, and E.A. Riskin. Mobileasl:: intelligibility of sign language video as constrained by mobile phone technology. In *Proceedings of the 8th International ACM SIGACCESS Conference on Computers and Accessibility*, page 78. ACM, 2006.

[24] CGSociety. Comparison of 3D tools. [Online] Available at http://wiki.cgsociety.o rg/index.php/Comparison_of_3d_tools, (Accessed: January 2010).

[25] L. Chittaro, F. Buttussi, and D. Nadalutti. MAge-AniM: a system for visual modeling of embodied agent animations and their replay on mobile devices. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, page 351. ACM, 2006.

[26] CMU Sphinx Project Team. CMU Sphinx project official website. [Online] Available at http://cmusphinx.sourceforge.net/wordpress/, (Accessed: April 2009).

[27] Comcast Team. Comcast official website. [Online] Available at http://www.comc ast.net/videomail/, (Accessed: April 2008).

[28] J. Connan. Integration of signed and verbal communication: South African sign language recognition and animation. [Online] Available at http://www.coe.uwc.a c.za/index.php/SASL.html, (Accessed: April 2010).

[29] S. Cox, M. Lincoln, J. Tryggvason, M. Nakisa, M. Wells, M. Tutt, and S. Abbott. TESSA, a system to aid communication with deaf people. In *Proceedings of the 5th International Acm Conference On Assistive Technologies*, page 212. ACM, 2002.

[30] E. Dolnick. Deafness as culture. *Atlantic Monthly*, 272(3):37–53, 1993.

[31] D.W.H. Doo. A subdivision algorithm for smoothing down irregular shaped polyhedrons. In *Proceedings of the Conference on Interactive Techniques in Computer Aided Design*, pages 157–165, 1978.

[32] R.M. Engelke, K. Colwell, and R.W. Schultz. Telecommunication device for the deaf with interrupt and pseudo-duplex capability, July 5 1994. US Patent 5,327,479.

[33] R.M. Engelke, K. Colwell, R.W. Schultz, J. Hilliard, and T. Vitek. Telecommunication device for the deaf with automatic transmission capability, July 11 1995. US Patent 5,432,837.

[34] R.M. Engelke, K. Colwell, R.W. Schultz, J. Hilliard, and T. Vitek. Telecommunication device operating under an enhanced TDD protocol, May 14 1996. US Patent 5,517,548.

[35] R.M. Engelke and K.R. Colwell. Telecommunications device with automatic code detection and switching, September 25 1990. US Patent 4,959,847.

[36] eSpeak Project Team. eSpeak project official website. [Online] Available at http://espeak.sourceforge.net/, (Accessed: April 2009).

[37] Eyejot Team. Eyejot official website. [Online] Available at http://www.eyejot.com/, (Accessed: April 2009).

[38] R. Farmer. Instant messaging—collaborative tool or educator's nightmare? In *Proceedings of the North American Web-based Learning Conference*, 2003.

[39] Festival Project Team. Festival project official website. [Online] Available at http://www.cstr.ed.ac.uk/projects/festival/, (Accessed: April 2009).

[40] S. Foster. Communication as social engagement: implications for interactions between deaf and hearing persons. *Scandinavian Audiology*, 27(4):116–124, 1998.

[41] J. Francik and P. Fabian. Animating sign language in the real time. In *Proceedings of the Conference on Applied Informatics*, pages 276–281. Citeseer, 2002.

[42] FreeTTS Project Team. FreeTTS project official website. [Online] Available at http://freetts.sourceforge.net/, (Accessed: April 2009).

[43] E. Giguere. The Information Module profile. August 2004. [Online] Available at http://developers.sun.com/mobility/imp/impintro/, (Accessed: January 2010).

[44] M. Glaser and W.D. Tucker. Telecommunications bridging between Deaf and hearing users in South Africa. In *Proceedings of the Conference and Workshop on Assistive Technologies for People with Vision and Hearing Impairments*. Citeseer.

[45] M. Glaser and W.D. Tucker. Web-based telephony bridges for the Deaf. In *Proceedings of the South African Telecommunications Networks and Applications Conference*, Wild Coast Sun, South Africa, 2001. Citeseer.

[46] S.M. Halawani. Arabic sign language translation system on mobile devices. *Computer Science and Network Security*, 8(1):251, 2008.

[47] A. Hameed. Information and communication technologies as a new learning tool for the deaf. *Relation*, 10(1.129):7856, 2009.

[48] H-Anim Working Group. H-Anim Working Group official website. [Online] Available at http://h-anim.org/, (Accessed: January 2010).

[49] J.L. Hernandez-Rebollar, N. Kyriakopoulos, and R.W. Lindeman. A new instrumented approach for translating American sign language into sound and text. In *Proceedings of the 6th IEEE International Conference on Automatic Face and Gesture Recognition*, pages 547–552, 2004.

[50] B. Hopkins. MIDP 3.0 features: Inter-MIDlet communication and events. October 2009. [Online] Available at http://java.sun.com/developer/technicalArticles/javame/midp3_enhance/, (Accessed: January 2010).

[51] S. Igi, M. Tamaru, Y. Yamamoto, M. Ujitani, and S. Sugita. Sign language synthesis for mobile environments. In *Proceedings of the 11th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*. Citeseer, 2003.

[52] D. Jacka, A. Reid, B. Merry, and J. Gain. A comparison of linear skinning techniques for character animation. In *Proceedings of the 5th International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa*, pages 29–31. Citeseer, 2007.

[53] Java Community Process Team. JSR 215: The Java Community Process version 2.6. [Online] Available at http://jcp.org/en/jsr/detail?id=215, (Accessed: June 2008).

[54] Java Community Process Team. JSR 913: The Java Community Process version 2.0. [Online] Available at http://jcp.org/en/jsr/detail?id=913, (Accessed: June 2008).

[55] Java Community Process Team. Java Community Process official website. [Online] Available at http://www.jcp.org/, (Accessed: January 2010).

[56] Java Community Process Team. JCP 2 procedures overview. [Online] Available at http://www.jcp.org/en/procedures/overview, (Accessed: January 2010).

[57] Java Community Process Team. JCP 2: Process document. May 2009. [Online] Available at http://www.jcp.org/en/procedures/jcp2/, (Accessed: January 2010).

[58] Java Community Process Team. JSR 135: Mobile Media API [Online] Available at http://www.jcp.org/en/jsr/detail?id=135, (Accessed: January 2010).

[59] Java Community Process Team. JSR 184: Mobile 3D Graphics API [Online] Available at http://www.jcp.org/en/jsr/detail?id=184, (Accessed: January 2010).

[60] Java Community Process Team. JSR 195: Information Module profile. [Online] Available at http://www.jcp.org/en/jsr/detail?id=195, (Accessed: January 2010).

[61] Java Community Process Team. JSR 242: Digital Set Top Box profile. [Online] Available at http://jcp.org/en/jsr/detail?id=242, (Accessed: January 2010).

[62] Java Community Process Team. JSR 30: Connected, Limited Device configuration version 1.0. [Online] Available at http://www.jcp.org/en/jsr/detail?id=30, (Accessed: January 2010).

[63] Java Community Process Team. JSR 30: Connected, Limited Device configuration version 1.1. [Online] Available at http://www.jcp.org/en/jsr/detail?id=139, (Accessed: January 2010).

[64] Java Community Process Team. JSR 36: Connected Device configuration. [Online] Available at http://www.jcp.org/en/jsr/detail?id=36, (Accessed: January 2010).

[65] Java Community Process Team. JSR 37: Mobile Information Device profile. [Online] Available at http://www.jcp.org/en/jsr/detail?id=37, (Accessed: January 2010).

[66] Java Community Process Team. JSR 82: Java APIs for Bluetooth. [Online] Available at http://www.jcp.org/en/jsr/detail?id=82, (Accessed: January 2010).

[67] M.W. Kadous. Machine recognition of Auslan signs using PowerGloves: Towards large-lexicon recognition of sign language. In *Proceedings of the Workshop on the Integration of Gesture in Language and Speech*, 1996.

[68] K. Karpouzis, G. Caridakis, S.E. Fotinea, and E. Efthimiou. Educational resources and implementation of a Greek sign language synthesis architecture. *Computers & Education*, 49(1):54–74, 2007.

[69] E. Keating and G. Mirus. American sign language in virtual space: Interactions between deaf users of computer-mediated video communication and the impact of technology on language practices. *Language in Society*, 32(05):693–714, 2004.

[70] R. Kennaway. Experience with and requirements for a gesture description language for synthetic animation. *Gesture-Based Communication in Human-Computer Interaction*, pages 449–450, 2004.

[71] Khronos Group. OpenGL ES official website. [Online] Available at http://www.khronos.org/opengles/, (Accessed: January 2010).

[72] Khronos Group. OpenGL official website. [Online] Available at http://www.opengl.org/, (Accessed: January 2010).

[73] J. Knudsen. Wireless Java: developing with J2ME. Apress, 2003.

[74] J. Knudsen. The Information Module profile. November 2002. [Online] Available at http://developers.sun.com/mobility/midp/articles/midp20/, (Accessed: January 2010).

[75] W.S. Lee and N. Magnenat-Thalmann. Virtual body morphing. In *Proceedings of the 14th Conference on Computer Animation*. Citeseer, 2001.

[76] R.H. Liang and M. Ouhyoung. A real-time continuous gesture recognition system for sign language. In *Proceedings of the International Conference on Automatic Face and Gesture Recognition*, pages 558–567, 1998.

[77] C. Loop. Smooth subdivision surfaces based on triangles. Master's thesis, Department of Mathematics, University of Utah, 1987.

[78] Z. Ma and W.D. Tucker. Adapting x264 to asynchronous video telephony for the Deaf. In *Proceedings of the South African Telecommunications Networks and Applications Conference*, 2008.

[79] MakeHuman Project Team. MakeHuman project official website. [Online] Available at http://www.makehuman.org/, (Accessed: April 2009).

[80] M.A. Mandel. ASCII-Stokoe notation: A computer-writable transliteration system for Stokoe notation of American sign language. [Online] Available at http://www .speakeasy.org/ mamandel/ASCII-Stokoe.html, (Accessed: January 2008).

[81] Microsoft Corporation. Microsoft Speech API official webpage. [Online] Available at http://www.microsoft.com/speech/speech2007/speechdevarticle.msp x, (Accessed: January 2010).

[82] MobiX3D Project Team. MobiX3D project official website. [Online] Available at http://hcilab.uniud.it/MobiX3D/, (Accessed: April 2009).

[83] S. Morrissey. Assistive translation technology for deaf people: translating into and animating Irish sign language. 2008. [Online] Available at http://doras.dcu.ie/15 199/1/Morrissey_icchp_08.pdf, (Accessed: April 2009).

[84] S. Morrissey and A. Way. An example-based approach to translating sign language. In *Workshop on Example-Based Machine Translation*, pages 109–116, Phuket, Thailand, 2005. Citeseer.

[85] L. Muir, I. Richardson, and S. Leaper. Gaze tracking and its application to video coding for sign language. In *Picture Coding Symposium*, pages 321–325. Citeseer, 2003.

[86] L.J. Muir and I.E.G. Richardson. Perception of sign language and its application to visual communications for deaf people. *Deaf Studies and Deaf Education*, 10(4):390, 2005.

[87] T. Mullen and T. Roosendaal. Introducing character animation with blender. Sybex, 2007.

[88] N. Naidoo. Gesture recognition using feature vectors. Master's thesis, University of the Western Cape, 2009.

[89] Y.J. Oh, K.H. Park, H. Jang, D.J. Kim, J.W. Jung, and Z. Bien. A development of sign language avatar for text to sign translator. In *Proceedings of the Korea International Conference on Intelligent Systems*, pages 1141–1145. Korea Institute of Intelligent Systems, 2005.

[90] Oracle Corporation. The Java Virtual Machine specification. [Online] Available at http://java.sun.com/docs/books/vmspec/, (Accessed: January 2010).

[91] E. Ort. FAQs: Java virtual machine and C virtual machine. March 2001. [Online] Available at http://developers.sun.com/mobility/configurations/questions/vmdif f/, (Accessed: January 2010).

[92] C.E. Ortiz. Summary of CLDC-based profiles. June 2006. [Online] Available at h ttp://developers.sun.com/mobility/midp/ttips/cldc/, (Accessed: January 2010).

[93] C.E. Ortiz. A survey of Java ME today. November 2007. [Online] Available at h ttp://developers.sun.com/mobility/getstart/articles/survey/, (Accessed: January 2010).

[94] F.I. Parke. Computer generated animation of faces. In *Proceedings of the ACM International Annual Conference*, volume 1, pages 451–457. ACM, 1972.

[95] S. Pasquariello and C. Pelachaud. Greta: A simple facial animation engine. In *Proceedings of the 6th Online World Conference on Soft Computing in Industrial Applications, Session on Soft Computing for Intelligent 3D Agents.* Citeseer, 2001.

[96] D. Pilling and P. Barrett. Text communication preferences of deaf people in the United Kingdom. *Deaf Studies and Deaf Education*, 13(1):92, 2008.

[97] D. Power, M.R. Power, and B. Rehling. German deaf people using text communication: Short Message Service, TTY, relay services, fax, and e-mail. *American Annals of the Deaf*, 152(3):291, 2007.

[98] M.R. Power, D. Power, and L. Horstmanshof. Deaf people communicating via SMS, TTY, relay service, fax, and computers in Australia. *Deaf Studies and Deaf Education*, 12(1):80, 2007.

[99] C. Rajah. Chereme-based recognition of isolated, dynamic gestures from South African Sign Language with hidden Markov models. Master's thesis, University of the Western Cape, 2006.

[100] Research Lab, Inc. Text To Wave ActiveX Client/Server DLL SDK v2.0 product page. [Online] Available at http://www.research-lab.com/texax03read.htm, (Accessed: January 2010).

[101] B.N. Riskin. Method and means for telecommunications by deaf persons utilizing a small hand held communications device, April 6 1993. US Patent 5,200,988.

[102] M. Sarbaugh-Thompson and M.S. Feldman. Electronic mail and organizational communication: does saying "hi" really matter? *Organization Science*, pages 685–698, 1998.

[103] H. Sasaki, T. Kuroda, Y. Manabe, and K. Chihara. Hit-wear: A menu system superimposing on a human hand for wearable computers. In *Proceedings of the International Conference on Artificial Reality and Teleexistence*, pages 146–153, 1999.

[104] T.W. Sederberg and S.R. Parry. Free-form deformation of solid geometric models. *ACM SIGGRAPH Computer Graphics*, 20(4):151–160, 1986.

[105] H. Seersb and M. Dyec. A perceptually optimised video coding system for sign language communication at low bit rates. *Signal Processing: Image Communication*, 21:531–549, 2006.

[106] V.M. Segers. The efficacy of the eigenvector approach to South African Sign Language identification. Master's thesis, University of the Western Cape, 2009.

[107] H. Seo and N. Magnenat-Thalmann. An automatic modeling of human bodies from sizing parameters. In *Proceedings of the Symposium on Interactive 3D Graphics*, page 26. ACM, 2003.

[108] SightSpeed Team. SightSpeed official website. [Online] Available at http://www.sightspeed.com/, (Accessed: April 2009).

[109] SignWriting Markup Language Team. SignWriting Markup Language official website. [Online] Available at http://sign-net.ucpel.tche.br/swml/, (Accessed: June 2008).

[110] Simon Project Team. Simon project official website. [Online] Available at http://simon-listens.org/, (Accessed: April 2009).

[111] C.C. Slama, C. Theurer, and S.W. Henriksen. Manual of photogrammetry. Falls Church, Virginia, 1980. American Society of Photogrammetry.

[112] Smith Mirco Software. Poser: Discover the art of 3D figure design. [Online] Available at http://my.smithmicro.com/mac/poser/index.html, (Accessed: November 2008).

[113] Sony Ericsson. Sony Ericsson C905 white paper. [Online] Available at http://developer.sonyericsson.com/cws/download/1/708/297/1260890274/100 360-wp_c905_cybershot_1.pdf, (Accessed: January 2010).

[114] T. Starner, J. Weaver, and A. Pentland. Real-time American sign language recognition using desk and wearable computer-based video. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(12):1371–1375, 1998.

[115] D. Stein, P. Dreuw, H. Ney, S. Morrissey, and A. Way. Hand in hand: automatic sign language to English translation. In *Proceedings of the Conference on Theoretical and Methodological Issues in Machine Translation*. Citeseer, 2007.

[116] C. Stobbart and E. Alant. Home-based literacy experiences of severely to profoundly deaf preschoolers and their hearing parents. *Developmental and Physical Disabilities*, 20(2):139–153, 2008.

[117] W.C. Stokoe. Sign language structure: An outline of the visual communication systems of the American deaf. *Deaf Studies and Deaf Education*, 10(1):3–37, 2005.

[118] D.J. Sturman. A brief history of motion capture for computer character animation. In *Proceedings of the ACM SIGGRAPH International Conference on Character Motion Systems*, 1994.

[119] Sun Microsystems, Inc. JavaSoft ships Java 1.0. [Online] Available at ht tp://www.sun.com/smi/Press/sunflash/1996-01/sunflash.960123.10561.xml, (Accessed: June 2008).

[120] Sun Microsystems, Inc. The AWT in 1.0 and 1.1. April 1999. [Online] Available at http://java.sun.com/products/jdk/awt/, (Accessed: January 2010).

[121] Sun Microsystems, Inc. Foundation profile overview. [Online] Available at http:/ /java.sun.com/products/foundation/overview.html, (Accessed: January 2010).

[122] Sun Microsystems, Inc. The K virtual machine (kvm). [Online] Available at http: //java.sun.com/products/cldc/wp/, (Accessed: January 2010).

[123] Sun Microsystems, Inc. Personal Basis profile overview. [Online] Available at http://java.sun.com/products/personalbasis/overview.html, (Accessed: January 2010).

[124] Sun Microsystems, Inc. Personal profile overview. [Online] Available at http://java.sun.com/products/personalprofile/overview.html, (Accessed: January 2010).

[125] V. Sutton. SignWriting official website. [Online] Available at http://www.signwriting.org/, (Accessed: April 2009).

[126] Systems Biology Markup Language Team. Systems Biology Markup Language official website. [Online] Available at http://www.sbml.org/, (Accessed: January 2008).

[127] Talking Text Team. Talking Text official website. [Online] Available at http://www.telstra.com.au/homephone/features_services/talking-text.html, (Accessed: April 2009).

[128] Telephone Interpreting Service for South Africa Team. Telephone Interpreting Service for South Africa official website. [Online] Available at http://www.dac.gov.za/projects/language_service.htm, (Accessed: April 2009).

[129] Telkom SA. Teldem product website. [Online] Available at http://www.telkom.co.za/attheoffice/products/phoneinstruments/cordlessphones/accessories/teldem/index.html, (Accessed: April 2009).

[130] S.C.L. Terra and R.A. Metoyer. Performance timing for keyframe animation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer animation*, pages 253–258. Eurographics Association, 2004.

[131] D. Thalmann, J. Shen, and E. Chauvineau. Fast realistic human body deformations for animation and VR applications. In *Proceedings of the 1996 Computer Graphics International Conference*. Citeseer, 1996.

[132] TokBox Team. TokBox official website. [Online] Available at http://www.tokbox.com/, (Accessed: April 2009).

[133] W. Tucker, M. Glaser, and J. Penton. A bridge for the problem of Deaf telephony. *Science in Africa*, 2002.

[134] TypeTalk Team. TypeTalk official website. [Online] Available at http://www.textrelay.org/, (Accessed: April 2009).

[135] University of Hamburg. HamNoSys official website. [Online] Available at http://www.sign-lang.uni-hamburg.de/projects/hamnosys.html, (Accessed: April 2009).

[136] University of Hamburg. HamNoSys version 3.0 official website. [Online] Available at http://www.signlang.uni-hamburg.de/Projekte/HamNoSys/HamNoSysErklaerungen/englisch/Contents.html, (Accessed: April 2009).

[137] University of Hamburg. HamNoSys version 4.0 official website. [Online] Available at http://www.signlang.uni-hamburg.de/Projekte/HamNoSys/HNS4.0/HNS4.0eng/Contents.html, (Accessed: April 2009).

[138] Vodacom. TalkingSMS official website. [Online] Available at http://www.vodacomsp.co.za/vspc/products/word_to_mouth.html, (Accessed: April 2009).

[139] VoxForge Project Team. VoxForge project official website. [Online] Available at http://www.voxforge.org/, (Accessed: April 2009).

[140] J.B. Walther and K.P. D'Addario. The impacts of emoticons on message interpretation in computer-mediated communication. *Social Science Computer Review*, 19(3):324, 2001.

[141] C. Wang, G. Wen, and J. Ma. A real-time large vocabulary recognition system for Chinese sign language. *Lecture Notes in Computer Science*, pages 86–95, 2002.

[142] A. Watt and M. Watt. Advanced animation and rendering techniques. Citeseer, 1992.

[143] J. White. An introduction to Java 2 Micro Edition (J2ME); Java in small things. In *Proceedings of the 23rd International Conference on Software Engineering*, page 725. IEEE Computer Society, 2001.

[144] J.R. Whitehill. Automatic real-time facial expression recognition for signed language translation. Master's thesis, University of the Western Cape, 2006.

[145] J. Wong, E.J. Holden, N. Lowe, and R. Owens. Real-time facial expressions in Auslan tuition system. In *Proceedings of the 5th IASTED International Conference on Computer Graphics and Imaging*. Citeseer, 2003.

[146] D. van Wyk. Virtual human modelling and animation for sign language visualisation. Master's thesis, University of the Western Cape, 2008.

[147] S. Yeates, E.J. Holden, and R. Owens. An animated Auslan tuition system. *Machine Graphics And Vision*, 12(2):203–214, 2003.

[148] L. Yi. Kiara: an open source SIP system to support Deaf telephony. In *Proceedings of the South African Telecommunications Networks and Applications Conference*, 2008.

[149] X. Zhu, J. Yang, and A. Waibel. Segmenting hands of arbitrary color. In *Proceedings of the IEEE International Conference on Automatic Face and Gesture Recognition*, page 446, 2000.

[150] L. van Zijl and D. Barker. South African Sign Language machine translation system. In *Proceedings of the 2nd International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa*, page 52. ACM, 2003.

[151] L. van Zijl and J. Fourie. Design and development of a generic signing avatar. In *Proceedings of the Conference on Graphics and Visualization in Engineering*, pages 95–100, Florida, USA, 2007.

[152] L. van Zijl and G. Olivrin. South African Sign Language assistive translation. In *Proceedings of the International Conference on Telehealth/Assistive Technologies*, pages 7–12. ACTA Press, 2008.

[153] I. Zwitserlood, M. Verlinden, J. Ros, and S. van der Schoot. Synthetic signing for the deaf: eSign. Mai, 2005.