VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui

Computer Science

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@mcs.vuw.ac.nz

Building an On-line New Zealand Sign Language Dictionary

Patrick Bray

Supervisor: Peter Andreae

Abstract

New Zealand Sign Language (NZSL) is the official language of the Deaf Community of New Zealand. NZSL is a gestural language consisting of messages conveyed through simultaneous movements of the hands, face and body. In the early nineties the Deaf Studies Research Unit (DSRU) at Victoria University of Wellington, undertook a project to collect information regarding all NZSL signs that were in use at that time. This information was collated and stored in a 4th Dimension Database, which has since become antiquated. This data consisted of artistic representations of the signs, a HamNoSys encoding and general information about the sign. As this data was too valuable a resource to be lost it was necessary to extract the data for storage in a more contemporary database system. The main objective of this project was to extract this data from the 4th Dimension Database and use it as a basis to build a web based NZSL dictionary. This dictionary was created using PostgreSQL for data storage and PHP (Hypertext Pre-processor) to dynamically produce the web pages. Future areas of development for this system include improving the user interface, the inclusion of streaming video footage of the signs and a virtual signing avatar.

Acknowledgements

First and foremost I would like to thank my supervisor Peter Andreae for all his help and guidance throughout this project.

I would also like to extend my gratitude to David McKee who was my sign language teacher at the beginning of this project and to the rest of the Victoria Universities' Deaf Studies Research Group for allowing me to get involved in such an interesting project.

Last of all I would also like to thank Roger Cliffe and Mark Pritchard for all the assistance that they gave me in setting up the various components of my project to work with the MCS system.

Table of Contents

1 History, Background and Motivation 1

1.1 Introduction to NZSL 1

1.2 History of the Project 1

1.3 NZSL Bill 2

1.4 Deaf Culture 2

1.5 Technology in Deaf Culture 3

1.6 HamNoSys 3

2 Project Outline 5

2.1 Project Objectives 5

2.2 Motivation 5

2.2.1 Why the 4th Dimension Database was no longer appropriate 5

2.2.2 Lack of NZSL Resources 5

2.3 Design Goals 6

2.4 Target Audience 6

2.5 Related Work 7

3 Technology Design Decisions 8

3.1 Database Management System 8

3.2 Comparison of PostgreSQL and MySQL 8

3.2.1 Cost 9

3.2.2 Reliability 9

3.2.3 Speed and Concurrency 9

3.2.4 Advanced Features 9

3.3 User Interface Technology 10

3.3.1 Motives for choosing PHP 10

3.4 Persistent vs. Non-Persistent Database Connections 10

3.5 PEAR Database Abstraction Layer 11

4 The Original Database 12

4.1 Original Database Structure 12

4.2 Original Database Data 14
4.3 Pictures 15
4.4 HamNoSys 15
5 Project Phases 15
5.1 Learning NZSL 16
5.2 Extracting the old Data 17
5.2.1 Database Security 18
5.2.2 Extraction Scripts 18
5.2.4 SIS and CVS Information 22
5.2.5 Pictures 22
5.2.6 Other Extraction Methods 23
5.2.6.1 ODBC 23
5.2.6.2 Pluggers Software's Postgres Plugin 24
5.2.6.3 4th Dimension Server and Sybase SQL 24
5.2.7 Results 24
5.3 Data Manipulation 24
5.3.1 Different Operating Systems 24
5.3.2 Referential Integrity Constraints 25
5.3.3 Data Entry Errors 25
5.3.4 Case Sensitivity 25
5.3.5 Maori & Foreign Language Words 25
5.3.6 HamNoSys 26
5.3.7 PDF Data Extraction 26
5.4 New Database Design 26
5.5 User Interface Design 28
5.6 Testing 29
5.6.1 User Interface Testing 29
5.6.2 User Acceptance Testing 30
5.6.3 Performance & Load Testing 30
5.6.3.1 Test Cases 31

5.6.3.2 Method 31
5.6.3.3 Results 31
6 Security 33
6.1 Authentication 33
6.2 Authentication Mechanisms 33
6.3 SQL Injection Attacks 36
6.4 Maintenance 38
7 Conclusions and Future Work 39
7.1 Conclusions 39
7.2 Future Work 40
7.2.1 User Acceptance Testing 40
7.2.2 Audit Trail 40
7.2.3 Image Optimization 40
7.2.4 Special Characters 40
7.2.5 Video Clips 40
7.2.6 User Interface 41
7.3 Future Extensions 41
7.3.1 Forum 41
7.3.2 Virtual Human Signing 42
7.3.3 Movement Specification 44
7.3.4 Mobile and PDA Access 44
Bibliography 45
Appendix A SQL Database Schema Definition 48
Appendix B 4D Database Structure 53

List of tables

Number *Page*

Table 4-1 Comparison of the HamNoSys encodings of right and left hand shapes for the sign computer 15

Table 5-1 Results of performance tests 32

Table 6-1 Hypothetical Database Table Dictionary_Users 36

Table 6-2 Use of SQLrand in preventing SQL injection attacks 38

List of figures

Number *Page*

Figure 1: Tele Type Writer 3
Figure 2: Basic HamNoSys character mappings (from Hanke and Schmaling, 2005) 4
Figure 3: 4th Dimension Database Schema 13
Figure 4: Artistic representation of the NZSL sign computer 15
Figure 5: HamNoSys encoding for the NZSL sign for "computer" 15
Figure 6: Comparison of English and NZSL Grammar. 16
Figure 7: Comparison of the NZSL signs for both heavy and light. 17
Figure 8: Example of simple NZSL conversation. 17
Figure 9: 4th Dimension procedure running in debug mode 18
Figure 10: Procedure used for the extraction of Data from the 4th Dimension Database . 19
Figure 11: Error message displayed when saving or printing from the 4th Dimension Database 19
Figure 12: Generic and customisable extraction procedure 20
Figure 13: Export dialog displayed as a result of running the procedure in figure 12 20
Figure 14 : Section of the 4D Database schema illustrating the use of sub-tables 21
Figure 15 : Example of the PDF that was created in order to extract data from the 4D Database 21
Figure 16 : Picture Extraction Script 23
Figure 17: Skeleton EER Diagram of the new Postgres Database 27
Figure 18: NZSL Dictionary Home Page 28
Figure 19: NZSL Dictionary Displaying all NZSL signs with an English Head Word beginning with an 'A' 29
Figure 20: Sign Details of the NZSL Sign for the Maori word whakairo (carving) 29
Figure 21: Component Diagram illustrating security vulnerabilities 34
Figure 22: Example CAPTCHA image 35
Figure 23: Original Database Query 37
Figure 24: Database Query after SQL Injection Attack 37

Figure 25: Database Query after SQL Injection Attack with input validation 37
Figure 26: Forum for discussion of aspects of NZSL and general Deaf Culture..... 42
Figure 27: Virtual Signer avatar "Visia 2" signing a television news item 43
Figure 28: HamNoSys representation of the German Sign Language sign "Going-To" .. 43
Figure 29: SiGML representation of the German Sign Language sign "Going-To" 44

1 History, Background and Motivation

1.1 Introduction to NZSL

New Zealand Sign Language is the language of the Deaf Community of New Zealand. It is a full language, with the same ability to communicate information as other languages, and its own vocabulary system and grammatical structure. It is a visual-gestural language, using simultaneous movement of the hands, face, and body to convey messages. It is important to note that NZSL is not simply a signed version of English nor is it just a collection of gestures but a full language.

It has only been in the past two decades that NZSL has come to be recognised as a real, natural, human language. In fact in the past, attitudes towards sign language were negative to the extent that many thought it was an inferior form of communication and that all intelligent Deaf children should instead master the oral system of lip reading. Furthermore in the early 1900's signing was actually forbidden at Sumner (now Van Asch Deaf Education Center), New Zealand's first school for the Deaf.

While New Zealand sign language is distinct from other sign languages it is a direct descendent of British Sign Language and is closely related to Australian Sign Language. Not only is New Zealand sign language distinct from other sign languages but there also exists different dialects that have developed through the student communities at New Zealand's four major schools for the Deaf.

As an indication of the scale of NZSL usage in New Zealand, figures from the 2001 census show that it is currently used by approximately 28,000 people. According to the Deaf Association an estimated 7,000 of these people are deaf (Deaf Association, 2005).

1.2 History of the Project

In 1991 work began at Victoria University into the gathering of information towards a Dictionary of New Zealand Sign Language. This process involved interviewing signers from all of the three major regions (Auckland, Wellington and Christchurch) and identifying the signs that they used and recognised. This information was then compiled and stored in a 4th Dimension Database. In 1998 Victoria's Deaf Studies Research Unit in conjunction with the Deaf association of New Zealand used this data to produce a Dictionary of NZSL.

This dictionary is similar in many ways to a normal bilingual dictionary as it translates between English words and NZSL signs; however there are some important differences.

The ordering of signs is based on properties of the sign rather than an alphabetical ordering and the NZSL entries involve drawings of the signs. As each sign must include a drawing the Dictionary is relatively large and the English to NZSL section is actually just an index with references to page numbers where the sign can be found. These factors mean that it is a rather difficult and expensive tool to use.

1.3 NZSL Bill

At present there is a Bill before Parliament to promote and maintain the use of New Zealand Sign Language by declaring New Zealand Sign Language as an official language of New Zealand. This Bill also aims to provide for the use of NZSL in legal proceedings and to introduce principles to guide government departments in the way in which they should make use of NZSL in the promotion of their services to the public (NZSL Bill, 2005).

With the introduction of the New Zealand Sign Language Bill into parliament earlier this year, NZSL is set to become the third official language of New Zealand, joining both English and Maori. At the first reading, on June 22nd 2004, the Bill was supported by all political parties. It was then referred to the Justice and Electoral Committee which reported back to the House on the 18th of July 2005. The Committee received 195 submissions in total on the NZSL Bill, of which all expressed overwhelming support. Due to the General Election in September 2005 the Bill is set to have a commencement date of the 1st March 2006.

As a result of this Bill, it is likely there will be more resources made available for projects in this area. Such projects could benefit dramatically from the data that was collected by the DSRU and is currently stored in the 4th Dimension database. However in its former state the data was of little use as it could not be easily accessed, updated or extracted.

1.4 Deaf Culture

Many of the users of this system are likely to come from the Deaf community. Therefore it is important that Deaf Culture is taken into consideration when building such a system. Deaf Culture in New Zealand consists of many unique protocols, traditions and celebrations.

For example people involved in the deaf community are traditionally given a sign-name as finger spelling long names can become cumbersome. These sign-names often reflect physical characteristics or personality traits of the signer. “In New Zealand it is

customary, when introducing oneself, to give one's name and sign-name with an explanation of the sign-name's origin" (Penman, 1999).

Members of the Deaf community have their own style of humour with Deaf jokes which are usually more visual then English jokes.

> "Deaf humour is very visual. Deaf jokes play on things that look similar, in comparison to English jokes, which play on sounds or play on words" (Victoria Manning, Mahony, 2003).

Deaf Culture has a strong foundation in Deaf Clubs and societies were deaf people have traditionally met for social and sporting activities such as deaf rugby and to maintain their strong social and cultural links.

1.5 Technology in Deaf Culture

Technology plays an integral part in the day to day lives of the deaf. Flashing lights in the house can announce a visitor at the door, an incoming telephone call or even a fire (Deaf Association 2005). Communication with family and friends is often done using faxes, computers and TTYs' (telephone typewriter devices) such as that illustrated in figure 1 below.

Figure 1: Tele Type Writer

This frequent use of technology lends itself to a culture of technological embrace and as such a system like this online sign language dictionary will most likely be met with enthusiasm.

1.6 HamNoSys

Developed at the University of Hamburg, HamNoSys (The Hamburg Sign Language Notation System) is a method of encoding the properties of a sign, including hand

shape, hand orientation, location, and movement (Hanke and Schmaling, 2005). HamNoSys is a detailed notation system similar in many ways to a phonetic transcription of an oral language and at present is one of the only mechanisms for the written representation of signs.

HamNoSys is comprised of approximately one hundred and fifty symbols indicating different properties of the sign. Examples of some of the basic hand-shape to character mappings are illustrated in figure 2 below.

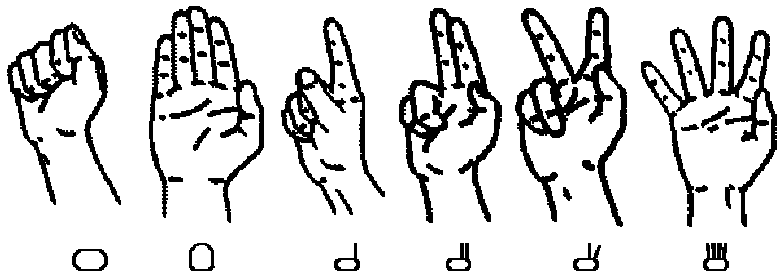


Figure 2: Basic HamNoSys character mappings (from Hanke and Schmaling, 2005)

2 Project Outline

2.1 Project Objectives

There were three main objectives of this project:

- To extract the relevant information from the now obsolete 4th Dimension database
- To design and build a new full strength SQL-based database to store the extracted data
- To design and build a set of interfaces into the data for the searching and updating of signs

2.2 Motivation

The main motivation behind this project was that the original 4D database was no longer being able to be easily used, as a result of this the data was in jeopardy of being lost. This data is too precious a resource to be lost and thus needed to be transferred to a more up to date database system.

2.2.1 Why the 4th Dimension Database was no longer appropriate

The 4th Dimension Database was no longer a feasible tool as it limited access to the one user logged into the machine on which it was stored. Furthermore the user interface forms for this database were designed for a portrait monitor and would not display correctly on a traditional landscape monitor.

This, coupled with the slow speed, poor user interface and irrelevant data made it a difficult and frustrating tool to use. As the original database designer could not be contacted and there was lack of the required expertise with the 4th Dimension DBMS, it was very difficult to support or extend the system.

2.2.2 Lack of NZSL Resources

At present there is a genuine lack of NZSL resources available. This contributes to the already strong social barriers that often prevent Deaf people from taking a more active role in society. In order to remove these barriers the Deaf community has

> "highlighted four priority areas that need immediate and long-term improvements. These areas are education, health, employment and public broadcasting." (Dyson, 2004)

Dyson goes on to allude to the need for NZSL resources such as this dictionary in her speech to the House during the first reading of the NZSL Bill:

> "To make information more accessible we need to translate written information into plain English, and use modern technologies, such as video conferencing and signed video clips on the internet and on television." (Dyson, 2004)

Through the creation of this online dictionary it is hoped to make information regarding NZSL more accessible to anyone learning or teaching NZSL.

2.3 Design Goals

This project was undertaken with three main design goals:

- Extensibility: This system should allow for future features, improvements and functionality to be easily incorporated.
- Maintainability: This system should require minimal maintenance.
- User Friendly: The system should be easy to use and require no previous knowledge of NZSL.

2.4 Target Audience

This system's target audience consists primarily of the members of the Victoria University DSRU. However it is also intended as a tool to be used by students of NZSL at Victoria and in the future, to be made available to the general public.

As the original NZSL dictionary was published by Bridget Willams Books they currently hold all publishing rights to this data in any form. Consequently the publication of any Sign Language Dictionary using this data would be a breech of these publishing rights. Therefore until these publishing rights can be acquired the only users of this Dictionary will be members of the DSRU and associated researchers.

2.5 Related Work

Presently the DSRU at Victoria University is conducting a project to capture video footage of all the NZSL signs. This footage will then be digitized and may, in the future, be incorporated into this database system and thus needs to be taken into consideration.

3 Technology Design Decisions

The correct choice of development tools plays an essential role in the success of any Information Technology project. Hence there were a number of important decisions that were made before the implementation of the system.

3.1 Database Management System

The first major design decision that needed to be made in this project was what DBMS to build the new database system in. In order to make this decision there were a number of criteria that needed to be evaluated. These criteria included the following

- Reliability: To ensure that the data would not be lost or be easily corrupted due to unforeseen circumstances.
- Speed: The chosen Database Management system should be able to run advanced queries quickly. This was important as the database is going to be used in an interactive web based environment where timing is of critical importance.
- Concurrency: The database should allow multiple users to be simultaneously accessing the same data and handle concurrent updates without placing the database into an inconsistent state.
- Scalability: The database management system needs to be able to cope with a large increase in the amount of both data and traffic in order to increase the extensibility of the system.
- Cost: The Database Management System should either be a low cost proprietary system or even better an open source solution.
- Advanced Features: The DBMS should support advanced database management features such as security to support in the extensibility of the system.
- Maintainability: As the system is to be used and maintained by members of the Deaf Studies Research Group who are not computer professionals the DBMS should require as little maintenance as possible.

3.2 Comparison of PostgreSQL and MySQL

Through the use of these criteria it was possible to narrow down two suitable Database Management Systems: PostgreSQL and MySQL

3.2.1 Cost

Both systems are open source solutions so are equal in terms of cost. However PostgreSQL is distributed under the Berkley Licensing agreement whereby developers are allowed to make use of Postgres in their application as long as a copy of the Berkley licensing agreement is also distributed. In contrast MySQL is distributed under the GPL license agreement, where in order to make use of the Database system within an application, the applications source code must also be distributed.

3.2.2 Reliability

Both MySQL and Postgres have large user base and open source development communities which makes them both attractive choices, with MySQL claiming to be "the world's most popular open source database" (MySQL 2005). This means that there is a large amount of technical and support documentation available for both DBMS'. The main factor influencing the decision to implement the database in Postgres however was its use in the past in the school of Mathematical and Computing Sciences (MCS) at Victoria University. This was important as the system is going to be hosted on the MCS network and supported by the school programming staff at least in the short term.

3.2.3 Speed and Concurrency

Independent Benchmarking has shown that MySQL out performs Postgres in terms of speed due to its minimalist nature (Perdue, 2000). These same benchmarks have also shown however that PostgreSQL is able to handle three times as many concurrent connections, as MySQL before generating any errors (Perdue, 2000). Furthermore when Postgres becomes overloaded, rather than failing it tends to degrade the performance of the system, whereas MySQL fails erroneously.

3.2.4 Advanced Features

MySQL aims to be a simpler database system providing "20% of the SQL Capabilities that are needed for 80% of database applications" (Jepson, 2001). However this somewhat limits the extensibility of the system. Postgres offers more of the advanced database management features than MySQL such as triggers, stored procedures, views and transactions.

Although more recent versions of MySQL have begun incorporating support for these features, they still lack support for the enforcement of referential integrity constraints, which is important in ensuring data consistency.

3.3 User Interface Technology

Another important design decision in this project was what technology to use to implement the user interface. In order to make the system as accessible as possible it was decided to implement a web based application. This means that users need only have a web browser installed to be able to make use of the system.

There were two main technologies considered for building the user interface these were sun Microsystems' JSP (Java Server Pages) and PHP. As the MCS web server where the dictionary is to be hosted is not a Microsoft IIS (Internet Information Services) Server, Microsoft's ASP (Active Server Pages) was not considered.

3.3.1 Motives for choosing PHP

PHP was chosen for a number of reasons. Like Java, PHP is a cross-platform solution that can be run on most variants of Windows, UNIX and Mac OSX. PHP has a large developer community supported by a plug-in based architecture. Third party scripts can easily be downloaded and their functionality incorporated into a web application saving valuable development time.

However, JSP outperforms PHP when PHP is running as a web server module (Titchkosky, Arlitt and Williamson, 2003). Running PHP through CGI causes even further performance degradations above those identified. This is because CGI performance is adversely affected by process creation (Apte, Hansen and Reeser, 2001), as for each new request that is received at the web server, a separate process must be created to serve the request.

Despite this problem, PHP was chosen for implementation of the Dictionary interface as it was already configured to run through CGI on the MCS Apache Web Server and comes installed by default on most web servers. This decision was also supported by an external developer at a meeting with the DSRU who may have further involvement in the project.

3.4 Persistent vs. Non-Persistent Database Connections

The reason that I chose to use non-persistent connections between the web server and the Postgres Database was as a result of the Dictionary system is being hosted on the MCS web server. This web server runs PHP as a CGI Wrapper and thus an instance of the PHP interpreter is created and destroyed for every page request. This means that any resources acquired by the interpreter such as database connections are closed at the

termination of the script. Therefore there would be no performance gain by using persistent connections. As the PHP interpreter has to check for open connections there could actually be a slight degradation in performance by using persistent connections.

3.5 PEAR Database Abstraction Layer

To increase the extensibility of this system in the future I decided to implement all Database access through the PEAR (PHP Extension and Application Repository) Database Abstraction layer. This abstraction layer provides features to allow PHP programs written for one Database Management System to function with all other supported DBMS's. This ensures that changes to the user interface will not be required if the data is moved to a different DBMS.

Unfortunately however this PEAR database library is not currently installed on the MCS web server. Due to this it was necessary to write my own PHP class that handled all access to the database, so any changes needed in regards to accessing the database would only need to be made in one file.

4 The Original Database

In order to understand why it was so important to extract the data from the old 4th Dimension database it is useful to examine the structure of the old database and the data that was stored within it.

4.1 Original Database Structure

By examining the original database schema as shown in figure 3 (See Appendix B for a larger version), it is evident that there are a number of areas where the design leaves room for improvement. For example the original 4th Dimension database did not conform to 1st normal form as it contained multi-valued attributes. One examples of these multi-valued attributes is illustrated by annotations i where a sign's usage could contain multiple values such as "offensive" and "vulgar".

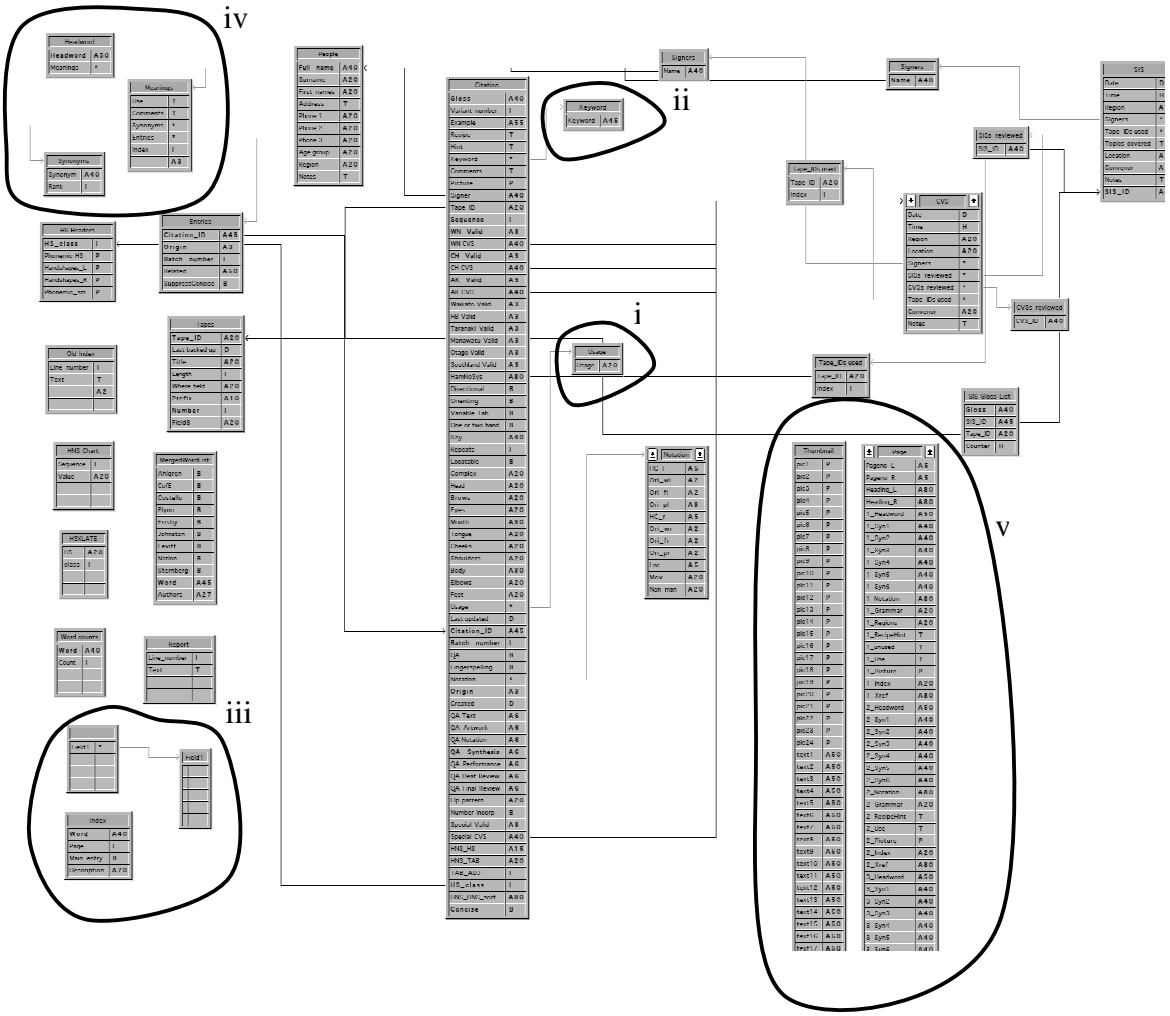


Figure 3: 4th Dimension Database Schema

A further example of these multi valued attributes is illustrated by annotation iv (shown in more detail in figure 14 on page 20). This area illustrates an extensive use of 4th Dimension sub-tables. 4D sub-tables are tables that are stored within a parent table and can only be accessed directly through the parent table. The relationship between the parent table and sub-table in 4th Dimension is a many to many relationship.

This is a very difficult and inefficient construct as access to any of the sub tables has to be made directly through the parent table. In the example, the 4th Dimension database has a multi-level parent sub-table relationship, with Meanings as a sub-table of Headword and both Entries and Synonyms as sub-tables of Meanings. This means that to find any synonyms of a given Meaning, the Headword table must first be traversed to find that meaning and then a nested query executed to find any associated synonyms.

This is a very difficult structure to program data retrieval and update queries for and has been simplified considerably by moving to a fully relational model.

As illustrated by annotation iii, on figure 3, the 4th Dimension database also had a number of unused temporary tables that were consuming valuable resources. Furthermore as the printed NZSL Dictionary was printed directly from this database it also contained a lot of irrelevant data such as that used for the formatting of the hard copy NZSL Dictionary as illustrated by annotation v.

As a consequence of this database not conforming to a relational model and a lack of data and referential integrity constraints, the data stored within the database had developed into an inconsistent state. This un-relational nature meant that it was difficult and inefficient to query the data structures and has been simplified a great deal by moving to a relational model.

4.2 Original Database Data

The data that was stored in the original 4th Dimension database can be divided up into five logical groups.

- Signer Information: Data related to the people who took part in the initial information gathering process.
- Session Information: Data directly related to the information gathering sessions, such as which signers took part, what VHS (Video Home System) video tape was used to record the session and the signs that were investigated.
- Sign Information: Details that were collated about each sign as a result of the information gathering process. There was a great deal of information stored about each sign, including when and in which context it should be used correctly and how it is signed.
- Conceptual Meaning Information: Information regarding how the NZSL sign meanings relate to the conceptual meanings associated with English words. This included information regarding the one English headword whose meaning was most similar to that of the NZSL sign, a definition of the meaning given by example English sentences and any other English words whose meanings were synonyms to the meaning of the NZSL sign. This involved a number of tables in a complicated web of linguistic relationships.
- Dictionary Information: Data that was used specifically to produce the paper-based, published NZSL dictionary. This included details of layout and the ordering of the signs within the dictionary.

4.3 Pictures

The original 4th Dimension database also contained a number of artistic representations of the NZSL signs. An example of one of these is shown in figure 4. These images were stored in an Adobe Illustrator format.

4.4 HamNoSys

The original 4th Dimension Database included a HamNoSys encoding for a number of the NZSL signs.

As an example the HamNoSys representation of the NZSL sign computer is displayed in figure 5. By comparing the HamNoSys string illustrated in figure 5 with the pictorial representation of the sign in figure 4 we can observe how the hand shape is represented by the iconic images shown in table 4-1. This shows the signers dominant hand (right hand in this picture) has all fingers extended and is bent which is illustrated by the horizontal bar in the HamNoSys character. The signer's recessive left hand is cupped with the thumb extending downwards. Dominant and recessive hands in NZSL are determined by the signers handedness

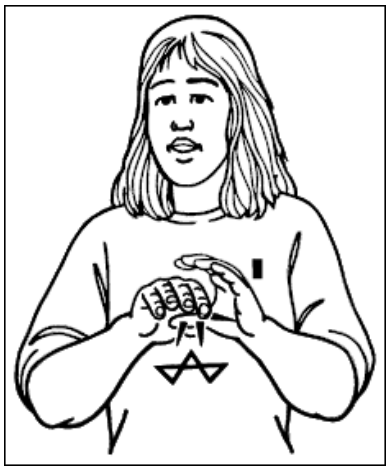


Figure 4: Artistic representation of the NZSL sign computer

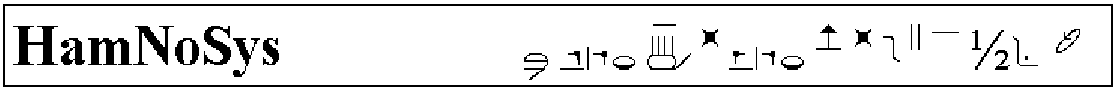


Figure 5: HamNoSys encoding for the NZSL sign for "computer".

Dominant Hand Shape	Recessive Hand Shape
	

Table 4-1 Comparison of the HamNoSys encodings of right and left hand shapes for the sign computerProject Phases

4.5 Learning NZSL

The first major component of my project involved participating in the DEAF 101 course offered by the School of Linguistics and Applied Languages at Victoria University. This is a beginners' course in NZSL which emphasises

> "the acquisition of basic receptive and expressive skills in sign language for every day communication and includes information about aspects of grammatical structure and Deaf community and culture". (VUW Deaf Studies, 2005)

This course was invaluable in aiding me to better understand a number of aspects of NZSL. These included the fact that the grammatical structure of NZSL differed from that of English and that signs may have different lexicon from one region to another. The difference between the grammatical structures of NZSL and English is best illustrated by figure 6 (from McKee and McKee, 2002).

Figure 6: Comparison of English and NZSL Grammar.

Another important aspect of NZSL that I was able to better understand through my studies in DEAF 101, was that NZSL signs involve more than just hand movements and that they are dependent on other body language such as the eyes, mouth and eyebrows. One prime example of this is illustrated below where the signs for both heavy and light are displayed.

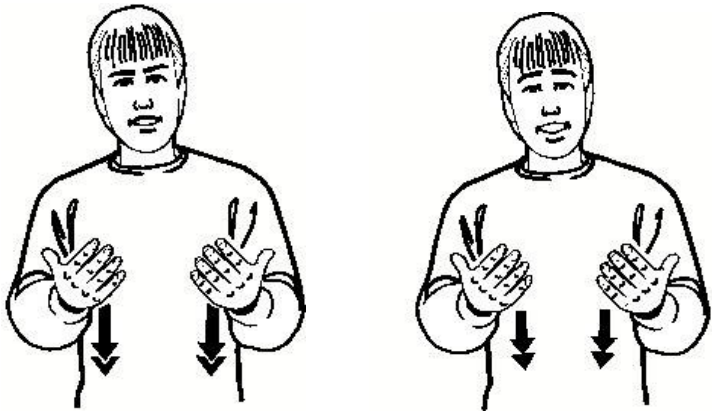


Figure 7: Comparison of the NZSL signs for both heavy and light.

Notice how similar the signs are with only a slight variation in the eyebrows and the speed of the bounce, yet these two signs have completely opposing meanings.

While completing this course was no easy task, having joined three weeks into the semester, I found it a thoroughly enjoyable and rewarding experience and am now able to carry out a somewhat primitive conversation in NZSL similar to that shown below.

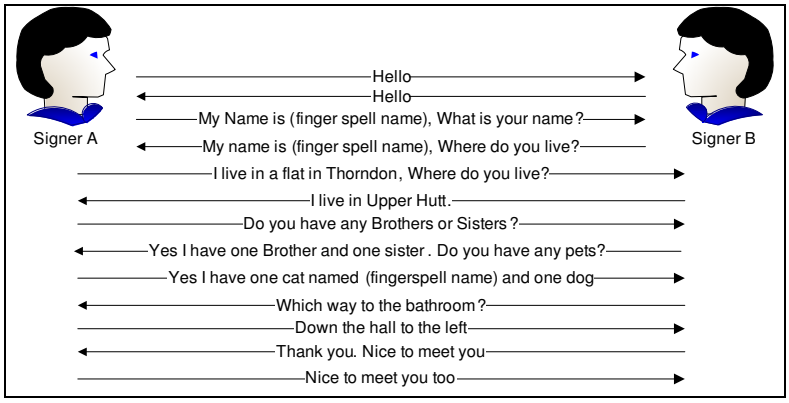


Figure 8: Example of simple NZSL conversation.

4.6 Extracting the old Data

The most important phase of this project involved the extraction of the relevant data from the original 4D Database. This was complicated by a number of factors and took much longer then anticipated.

4.6.1 Database Security

The original Database designer had enforced tight security constraints on the database as to what the users were authorised to access. This made it difficult to enter the database design mode and therefore limited the tools available for the extraction. The only available option for Data extraction in user mode was to print reports to PDF files. As the original database designer was no longer contactable and we had no administrator password this posed a significant problem.

In order to solve this problem we found a solution where it was possible to force a procedure loaded on one of the form events to run in debug mode. This was achieved by pressing and holding the control, alt and apple keys in order and then double clicking the form. This is illustrated in figure 9 below. Once this procedure was running in debug mode it was possible to edit the procedure. This allowed us to enter the database design mode without requiring an administrator password.

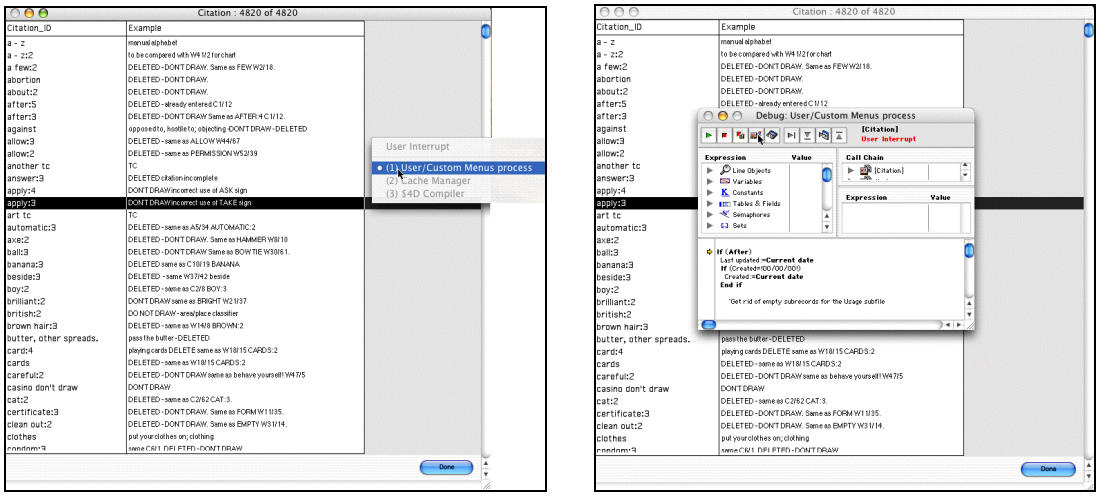


Figure 9: 4th Dimension procedure running in debug mode

4.6.2 Extraction Scripts

Having access to 4th Dimension's design mode enabled the writing of specific extraction procedures that could be loaded onto the existing form events. Unfortunately 4th Dimension does not support SQL (Structured Query Language) as a data manipulation Language and instead provides its own scripting language.

The 4D scripting language is a very extensive Pascal-like language with a rich library of functions and control structures. However due to the obsolete nature of the 4th Dimension database system there was an inherent lack of API (Application Programming Interface) resources available. Although we were able to source a copy of the original HTML (Hyper Text Mark up Language) based 4th Dimension Language Reference that was supplied with the Database. This enabled me to write the below procedure for the specific output of some of the data into delimited text files:

```
If(Form Event = 4)
     OutputForm([Table],"StandardOut"); //Set Output Form
     FldDelimit = 35; //ASCII Code for Field Delimiter
     RecDelimit = 36; //ASCII Code for Record Delimiter
     ExportText([Table],"./Filename.txt"); //Export Data
End if
```

Figure 10: Procedure used for the extraction of Data from the 4th Dimension Database

As illustrated by the procedure above it is necessary in 4th Dimension to set the Output Form for the table of which the data is to be extracted. This is because 4th Dimension will only output the fields that are displayed on the currently loaded form for that table. This in it self became a problem as a number of the data tables did not have an associated form which contained all of the data fields.

Furthermore it was not possible to simply create a new form with all the data fields, as when attempting to save this new form the 4th Dimension database runtime would crash displaying the vague OSX error illustrated in figure 11 below:

Figure 11: Error message displayed when saving or printing from the 4th Dimension Database

Fortunately I discovered another more generalised method that could be called, for the extraction of the data from the table files as illustrated in the procedure below:

```
If (Form Event = 4)
          Extract Data ("Table";*;); //Display Export Dialog
End if
```

Figure 12: Generic and customisable extraction procedure

Executing the above code would display a Data Export dialog box like that shown in figure 13. Using this dialog it was possible to select which fields to export, the delimiters to use and the format of the output file. This was the method that was used to export the majority of the Data from the 4th Dimension Database.

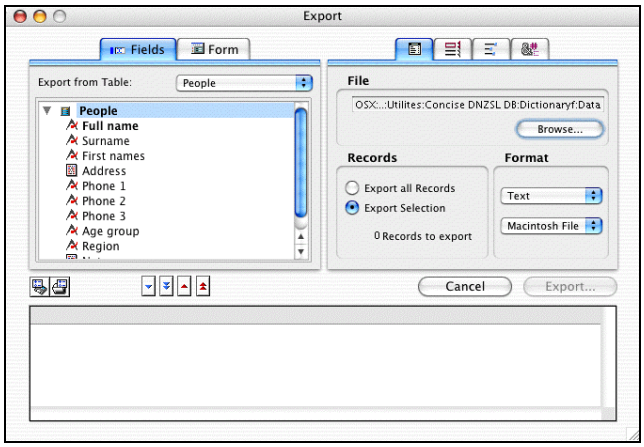


Figure 13: Export dialog displayed as a result of running the procedure in figure 12

4.6.3 Sub Tables

Unfortunately 4th Dimension does not adhere to a purely relational model. One of the unusual "features" that it offers is sub-tables, tables that are associated with and only accessible through their parent tables. This meant that it was not possible to use the aforementioned output dialog method to extract data from tables that had associated sub-tables, without loosing the information stored within these sub-tables and the links to the parent tables.

This was a significant problem as the original 4th Dimension Database contained a number of these sub-tables as illustrated in figure 14 below:

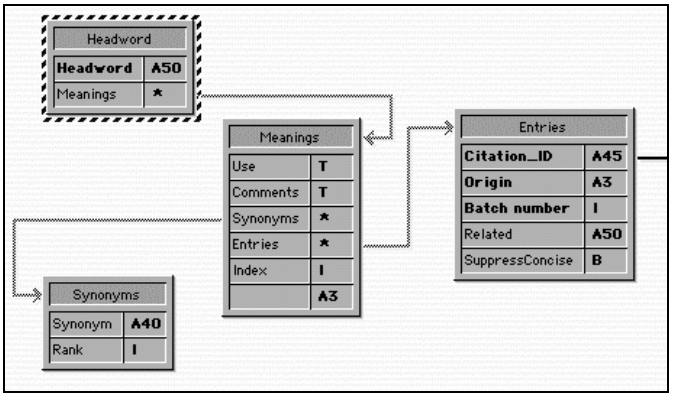
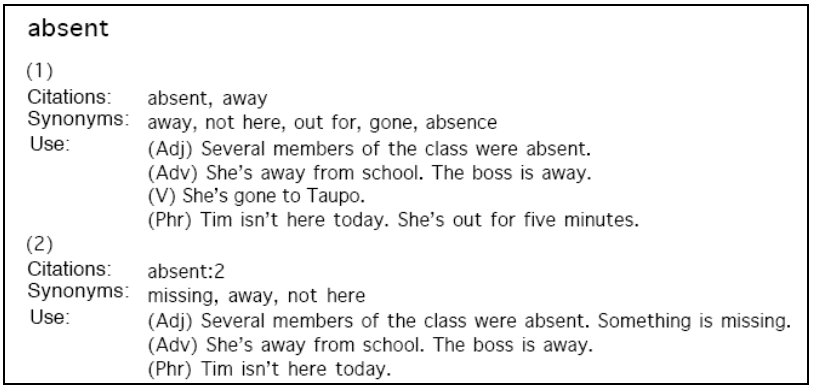


Figure 14 : Section of the 4D Database schema illustrating the use of sub-tables

Figure 14 above shows that Meanings is a sub-table of Headword which in turn has two sub-tables; Entries and Synonyms. These four tables contain all the data pertaining to how the NZSL signs relate to English words. Therefore it was vital that this data be extracted.

To solve this problem it was necessary to use one of the existing printing procedures, which iterated through the parent table and its associated sub-tables and created a report containing all this information. We were then able to print this report to a PDF (Portable Document Format) printer and thus create a PDF copy of the report. A sample of the PDF document that was created is shown below:



absent

(1)

Citations: absent, away
Synonyms: away, not here, out for, gone, absence
Use: (Adj) Several members of the class were absent.
(Adv) She's away from school. The boss is away.
(V) She's gone to Taupo.
(Phr) Tim isn't here today. She's out for five minutes.

(2)

Citations: absent:2
Synonyms: missing, away, not here
Use: (Adj) Several members of the class were absent. Something is missing.
(Adv) She's away from school. The boss is away.
(Phr) Tim isn't here today.

Figure 15 : Example of the PDF that was created in order to extract data from the 4D Database

It was then possible to extract the relevant data from this PDF file and manipulate it into a format that was able to be stored inside the Postgres Database. This will be explained in greater detail in section 5.3 Data Manipulation.

4.6.4 SIS and CVS Information

Unfortunately however there was no stored procedure available to print out all the records of the CVS or SIS tables which also contained sub-table relationships. This meant while it was possible to extract the majority of the data from these tables using the export dialog method, it was not possible to extract the relationships between these tables.

As these tables contained information regarding how the dictionary information was collected and collated, rather then information directly related to the dictionary, it was deemed a low priority and will need to be extracted in the future. This extraction could be done by writing an extraction procedure modelled on that used for the extraction of the NZSL – English correspondence information and then printing this to a PDF similar to that discussed in 5.2.3. The tables for this data to be imported into have in fact been designed and created but will need to be populated once this data has been extracted.

4.6.5 Pictures

The original 4th Dimension database also stored a number of artistic drawings for a subset of all the signs. These drawings were stored in an Adobe Illustrator format. Investigation has shown that this Adobe Illustrator format is in essence a postscript image file with some missing postscript packages. This meant that it was possible to write a shell script to add these packages back into the original image to create a postscript file. Using shell scripts this file could then be converted into a browser friendly format such as jpeg (Joint Photographic Expert Group), gif (Graphic Interchange Format) or png (Portable Network Graphics).

However, the only format in which it was possible to extract these drawings from the 4th Dimension database using the aforementioned Export Dialog method, was a proprietary 4th Dimension file format. Furthermore once a file containing these pictures had been extracted it would then be necessary to remove the 4th Dimension file format information and convert these images as mentioned above. As there was no clear definition as to where each picture began and finished it was not possible to automate this process and thus this method of extraction was abandoned.

The next extraction approach taken was to write the procedure in figure 15 to extract each picture to a picture file. While this procedure was somewhat successful as I was able to export a picture to a file and then view it, it was not possible to programmatically specify a filename. This meant that in order to extract the pictures using this method I would have had to specify a filename for each. This was not possible as there were 4820

signs in total stored in the database. In addition there was no way to find out which sign the current picture related to when specifying this filename.

```
var OutputHandle
var OutputFileName
If (Form event=4)
     ALL RECORDS([Citation])
     While (Not(End selection([Citation])))
          OutputFileName = [Citation]Citation_ID
          OutputHandle:=Create document(OutputFiSleName;"PICT")
          SAVE PICTURE TO FILE(OutputHandle;[Citation]Picture;)
          CLOSE DOCUMENT(OutputHandle)
          NEXT RECORD([Citation])
     End while
End If
```

Figure 16: Picture Extraction Script

Before the original database designer left he outputted the contents of the 4th Dimension database for Peter Andreae. This output also contained copies of the pictures. After investigation I found that these pictures were actually a complete set of exact copies of those stored in the database.

This investigation involved outputting a list of filenames and using regular expression transformations to remove the extra formatting from the sign identifiers that were stored in the filename. It was then necessary to write a small java program to parse this list and compare it against the complete list of all sign identifiers, which had associated pictures. Using this java program I found that a number of these files that Peter had were incorrectly named and there were a small number of missing pictures which needed to be manually extracted from the 4th Dimension Database.

4.6.6 Other Extraction Methods

There were a number of other methods identified for the extraction of the data from the 4th Dimension Database. However most of these required a lot of additional funds to purchase third party software or required a more up to date version of 4th Dimension.

4.6.6.1 ODBC

One of these methods was to create an ODBC (Object Database Connectivity) connection to the 4th Dimension database and write a program to execute queries to extract the data. This however required 4th Dimension Server Edition, which retails at $1600.

4.6.6.2 Pluggers Software's Postgres Plugin

Another of these methods involved purchasing a third party 4th Dimension plug-in which would make it possible to simply clone the 4th Dimension Database into a Postgres Database. This plug-in however costs $300 and requires 4th Dimension MAC OSX Version 10.3 and up. The 4th Dimension version that was utilised in this project was 7.0.2.

4.6.6.3 4th Dimension Server and Sybase SQL

Another method that was discovered to extract data from a 4th Dimension Database would use 4th Dimension Server Edition and Sybase SQL. Using this method it would be possible to write a 4th Dimension Procedure to create a connection to a Sybase Database and then write SQL Queries to extract the data from the 4th Dimension database into associated tables in the Sybase Database. As the Server version of 4th Dimension retails for $1600 and Sybase for $800, this extraction method was not further considered.

4.6.7 Results

Through these various methods, the majority of the relevant data was extracted into a collection of delimited text files. These text files were then manipulated into a suitable format and inputted into the new Postgres database as outlined in section 5.3. The images were converted to PNG files and their filenames changed to reflect the unique identifier of the sign that they represent. Following this the image files were moved to a single directory so that they could be dynamically loaded by the user interface.

4.7 Data Manipulation

Once the data had been extracted from the 4th Dimension Database, it needed to be manipulated into a format that could be stored in a Postgres Database.

4.7.1 Different Operating Systems

As the 4th Dimension Database was stored on a MacOSX operating system, there were differences in the data representation. In order to fix this, files created on the MacOSX architecture had to be run through a script to translate end-of-line characters from Carriage Returns (CR) that are used on Mac to Line Feed's (LF) that are used in Unix. This needed to be done before any other scripts could be run on these text files to do things such as sort the records and remove duplicates.

4.7.2 Referential Integrity Constraints

Version 7.0.2 of 4th Dimension did not support referential integrity constraints. This meant that when attempting to enter the data extracted from the 4th Dimension Database into the new Postgres database there were a number of referential integrity violations.

Examples of such referential integrity violations included information regarding one of the signers (Claire Holtham) who was married to Matthew Holtham during the original data collection phase. Claire's details were stored against her maiden name in one table and stored with her married name in another. In this instance it was possible to infer that these two records related to the same person, through comments detailed in one of the records.

4.7.3 Data Entry Errors

As the original database did not enforce referential integrity constraints, there were also a large number of spelling and data entry mistakes in the data that was extracted from the 4th Dimension Database. This required a lot of time to manually compare records between tables to see where the errors existed and to attempt to infer the correct format for the data.

4.7.4 Case Sensitivity

As Postgres is case sensitive and 4th Dimension is not there were also problems associated with referential integrity constraints that were placed upon text fields. In order to solve this problem it was necessary to use regular expressions to manipulate the text files so that the use of capital letters would be consistent. It was important however to ensure that none of the meaning of the data was lost through this manipulation, for example the removing capital letters from nouns.

4.7.5 Maori & Foreign Language Words

Problems also arose from the use of special characters for Māori and foreign language words. This would not have been such a problem if the use of these characters was consistent, however the 4D database contained instances where the same word would be stored with an accent in one table and without in another. The short term solution to this was to simply remove all these special characters. The correct solution which will need to be carried out over summer and will require a lot more time is to manually match these characters up with those which are printed in the NZSL Dictionary.

4.7.6 HamNoSys

The HamNoSys encodings stored in the dictionary also posed a significant problem to extract. This HamNoSys encoding is implemented in a font package that is available in both true type format (ttf) and as an apple font.

The problem that arose was that the data in the original 4th Dimension database was encoded using Version 2 of the HamNoSys font. The current version, version 4 is the only version that is available for download from the University of Hamburg's website. Since version 2 there have been some minor changes to the mapping of characters above ascii 127. From correspondence with Thomas Hanke from the University of Hamburg it was found that these mappings should only affect encodings that were made using a windows version of the font. The reason that the HamNoSys was not displaying correctly was that this font was not installed when the data was extracted. This meant that it was necessary to re-install this font on the MACOSX machine and then re-extract all of the HamNoSys encodings.

4.7.7 PDF Data Extraction

In order to extract the English to NZSL correspondence information from the database it was necessary to print this information to a PDF file. A sample of the format of this PDF is displayed in figure 14. Converting this data into a format that could be read into Postgres using SQL commands involved using regular expression transformations and Macro programs written in emacs.

Once this data had been converted into a machine readable format, I was able to write Java programs to parse this data. These java programs were used to create the specific SQL COPY commands for each of the four tables whose data was stored in the PDF file. Once the SQL commands had been constructed, the data was entered into the Postgres Database and any Database Constraint violations were manually resolved.

4.8 New Database Design

When designing a new Postgres database to store data extracted from the 4th Dimension database, it was necessary to retain much of the original structure. This was to ensure that none of the original meaning of the data was lost. It was possible however to improve upon the original database structure by adding a number of constraints to the data to ensure its validity and consistency.

Example of these constraints included referential integrity constraints between tables, such as Signer and SIS (Signer Information Session) where, in the old system it was possible to have a Signer Information Session without a participating signer. As 4th Dimension did not support referential integrity constraints there were a number of data inconsistency problems encountered when these constraints were added which needed to be manually resolved.

One area where the original schema was significantly modified however was the tables which stored information regarding the NZSL – English correspondence. As mentioned earlier these tables were implemented as a single table Headword with associated sub tables. These sub tables were redesigned from the non-relational structure in figure 14 to a fully relational model illustrated in figure 17.

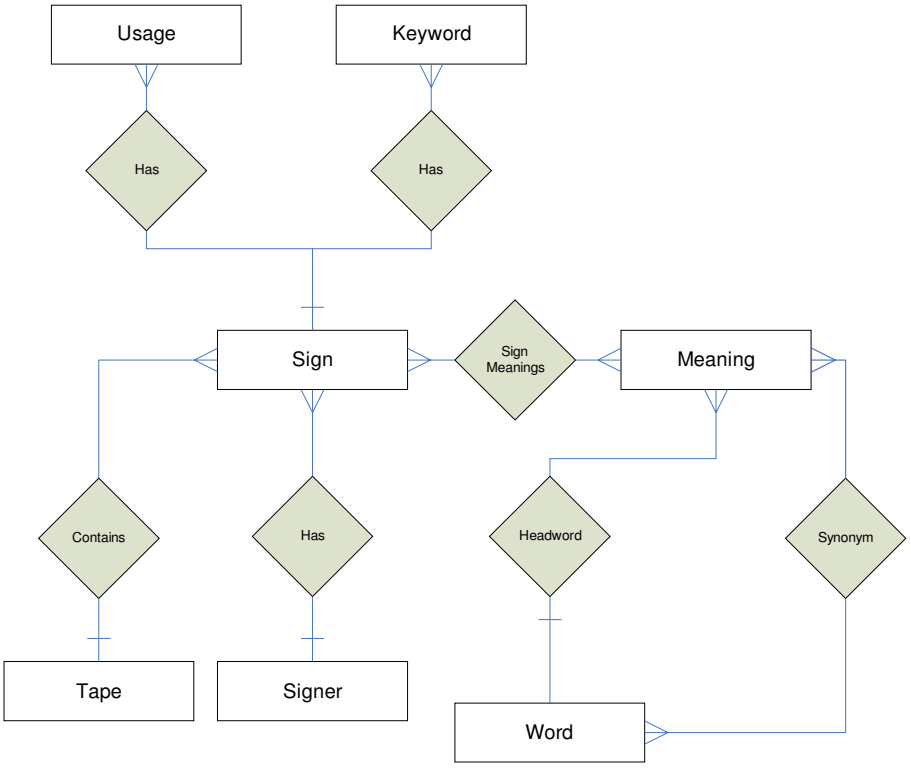


Figure 17: Skeleton EER Diagram of the new Postgres Database

After designing and constructing this new Relational Postgres Database the extracted Data had to be inserted. This was a very arduous task due to a number of data

inconsistencies which had to be manually resolved. A more complete outline of the database structure with the SQL schema definition can be found in Appendix A

4.9 User Interface Design

The two main design goals in building the user interface for the Dictionary was to make it as simple and functional as possible whilst maintaining extensibility. This simplicity is illustrated in figure 18 where the NZSL Dictionary Homepage is displayed with links to the different search types available and the ability to login, in order to update the signs. Each of these links relates directly to one of the specific use cases that were identified in the system requirements phase.

Figure 18: NZSL Dictionary Home Page

Due to the limited timeframe available for this phase of the project, the main goal in building the user interface was a working prototype for a future implementation. This meant that only a limited amount of the overall functionality was implemented. The current user interface has the ability to search and update signs based on a wide range of criteria but not particularly easily.

One of these searches is illustrated in figure 19 where the user is browsing for a sign based on the alphabetic ordering of the English Headword (English word whose meaning is most like that of the NZSL sign).

The sheer volume of data that is stored in the dictionary about each sign made it difficult to design simple user interface forms. To solve this problem I broke up the information stored about each sign into specific groups. Examples of such groupings included general sign information *vs.* information regarding the location of where the

sign originated from and was used. Figure 20 illustrates some of the information that is stored relating to the NZSL sign for the Maori word whakairo (carving).

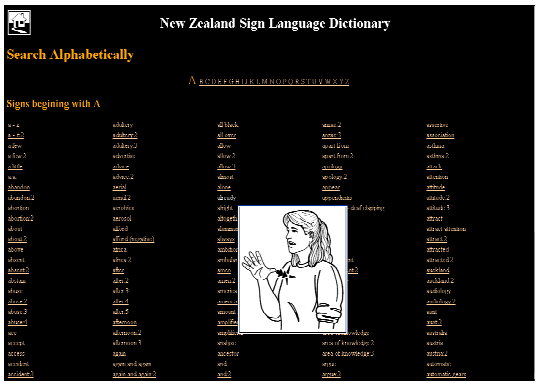


Figure 19: NZSL Dictionary Displaying all NZSL signs with an English Head Word beginning with an 'A'

Figure 20: Sign Details of the NZSL Sign for the Maori word whakairo (carving)

In order to ensure that users with disabilities such as those who have vision impairments are able to easily make use of the Dictionary, the Interface was designed with extensive use of CSS (Cascading Style Sheets). The intention of this was to allow users to specify their individual preference as to which style is displayed.

Suffering from colour blindness myself this is something which I can easily relate to. My colour blindness also means that the present colour scheme of the prototype user interface will most likely be changed or altered to the preferences of the DSRU, the use of CSS will make this a very straightforward change.

4.10 Testing

4.10.1 User Interface Testing

Once I had designed and built the user interface, it was important to test that it was displayed correctly under a number of different conditions. These conditions included the web browsers that the client was using and versions of these browsers, the Operating System that was running, the screen resolution and having both cookies and JavaScript disabled.

Testing these factors involved performing five of the major use cases that were identified.

- Searching for a particular sign based on the English equivalent
- Browsing signs by alphabetical order
- Searching for a sign based on properties of the sign
- Updating a sign
- Looking up the meaning of a sign and its synonyms
- Authenticating the user

These test cases were performed using the most popular web browsers as illustrated below under the three main Operating System variants Windows (Windows Server 2003), UNIX (NetBSD) and Apple's MAC OSX.

- Mozilla Firefox Version 1.0.6
- Mozilla Version 1.7.11
- Microsoft Internet Explorer Version 6.0
- Microsoft Internet Explorer for MAC Version 5.2
- NetScape Navigator 7.1
- Konqueror Version 3.4.2

These tests were also carried out under the following screen resolutions 1280x1024 and 800x600. This was taken into consideration in order to enable the system to be used by people who may suffer from vision impairments and need to have their screen resolution set lower and also to accommodate for users who had older equipment.

These tests resulted in the NZSL Dictionary being displayed correctly and usable in all tests with only negligible differences in terms of the look and feel of the site.

4.10.2 User Acceptance Testing

It was hoped to conduct usability and user acceptance tests of the dictionary with members of the DSRU on completion of the prototype interface. However due to the time constraints and the difficulty of scheduling such meetings, this has not been possible to date but is expected to be completed over next couple of weeks when the system is delivered.

4.10.3 Performance & Load Testing

An important consideration in any web-based system is the ability to handle unusually high or peak levels of load whilst retaining system performance. While it was not possible to stress test the MCS web server as it also hosts a number of other web sites it

was possible to run some tests in order to get insight into the time taken to perform some of the database queries executed regularly in the system.

4.10.3.1 Test Cases

These tests included

- Creating connections to the database
- Performing a simple select of all the data fields in a single randomly selected sign
- Performing a simple update to one of these signs
- Performing the optimal joining query required to display the meaning of a word its NZSL sign(s) and any synonyms
- Performing a non-optimal joining query to display the meaning of a word its NZSL sign(s) and any synonyms
- Searching for a sign based on the English equivalent

4.10.3.2 Method

All the above tests except those run on the 4th Dimension database were run 100,000 times on five separate occasions to take into account the variability of network and server load. The 4th Dimension database tests used as a comparison were tested using a smaller sample as these tests had to be performed manually. Results of these tests are summarised in table 5-1.

4.10.3.3 Results

In order to test the performance gain that could be achieved if PHP was implemented as a module on the web server rather than running through CGI, I decided to test the average time taken to create a connection to the backend database. This average after 100,000 iterations taken on five separate occasions was 21ms. In comparison the difference between the time taken to perform an optimized vs. non-optimized query was 38ms. This shows that while there would be a significant performance gain by introducing persistent database connections, a larger performance gain can be achieved by ensuring that all database queries that are executed are optimal and that only the required fields and records are returned.

Test	Average	Standard Deviation	Minimum	Maximum
Optimal Query	14 ms	7 ms	14 ms	302 ms
Non-Optimal Query	52 ms	13 ms	49 ms	358 ms
Simple Update	26 ms	9 ms	18 ms	106 ms
Creating Connections	21 ms	5 ms	19 ms	276 ms
Search	140 ms	24 ms	126 ms	530 ms
4D Search	99 s	4.4 s	92 s	107 s

Table 4-2 Results of performance tests

In order to test the speed of updating signs in the Database, I tested how long it took to update all the fields of a single record. As updates to the NZSL Database are likely to be a lot less frequent than database reads this was not as important a test. It would also have been good to test multiple concurrent connections to the database attempting to update the same record. This would test the concurrency control and record contention but this was not so important due to the static nature of the data in the database and the small number of users who will have write access to the database.

As a base case for comparison I tested the time it took to search for a sign in the old 4th Dimension Database. The average time taken to search for a sign when the input was not matched exactly was 99s. In comparison the average time taken to search for the same sign in the new Postgres Database was only 140ms. This is a remarkable improvement especially considering that the original 4th Dimension Database search would often not return any results and once started the search could not be manually stopped by the user.

5 Security

As the DSRU does not currently hold the publishing rights to the data used in this dictionary it was important to restrict access to the system. Furthermore as this data is such a valuable resource it is important that its integrity is not compromised by malicious users.

5.1 Authentication

In order to ensure the validity of the data stored within the database, it was important to establish constraints on which users were authorised to update the database and what information they were authorised to access.

There are at least four potential user groups;

- Administrators: That should have full access to the database and the PHP source code used to build the interfaces. This group included myself, Peter Andreae and the MCS programmers
- Members of the DSRU: who should be able to update information in the database using the PHP based graphical user interface
- Students and Staff from Victoria University: due to the copyright on the pictures held within the database these users need to be separated from the General public
- Normal Users: Members of the general public

5.2 Authentication Mechanisms

I investigated two authentication mechanisms for handling the authentication of users within the dictionary system.

The first mechanism involved storing data about the users within the Postgres database itself. In order to implement this system the database needed to store a tuple (username, encrypted password, and group) about each user. Using this information, the user interface would display a login form where users would enter their username and password. The password would then be encrypted using the same one way encryption algorithm (such as an md5 hash) as used to encrypt it for the database. The resulting value is then compared with the encrypted password stored in the database. If it matches then this user would be authenticated and granted the access permissions assigned to that group.

While this means that the password is not stored in plain text there are still a number of security flaws. One such flaw in this system is that the password would be transmitted across the network from the client to the server in plain text and would therefore be vulnerable to packet sniffing. To solve this problem the web server on which the PHP interfaces were running would need to implement SSL (Secure Socket Layer) to create a secure connection between both server and client.

Furthermore as the Web server and Database server are likely to be hosted on separate machines there is the problem of replay attacks. This is illustrated in figure 21 below where an attacker could intercept packets from the web server containing the encrypted version of the password and later use this to gain access to the database. To prevent this from occurring there would need to be a temporary nonce value added to the authentication between web server and database server. This however is not such an important problem, as the communication between the Web Server and Database server would be carried out within the trusted demilitarized zone. Therefore the threat of malicious intruders should be significantly decreased through the use of a firewall.

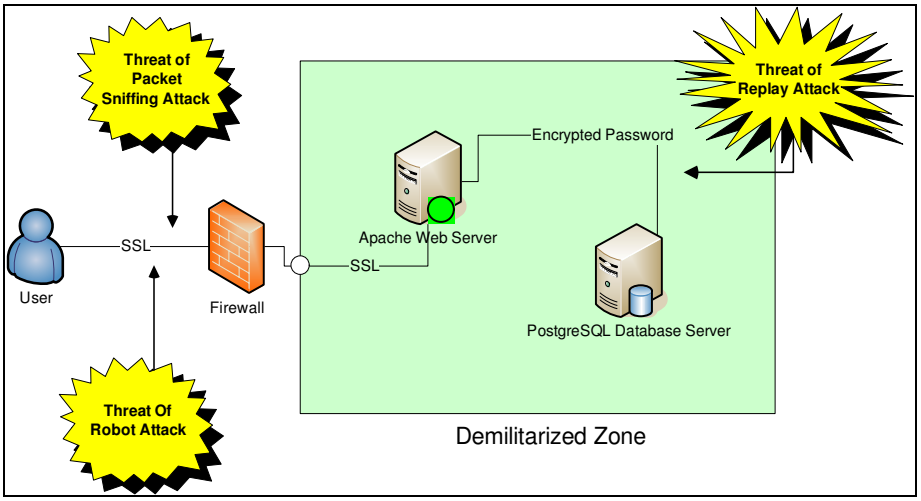


Figure 21: Component Diagram illustrating security vulnerabilities

Another problem with this solution is that it is possible for malicious users to write Robots (automated programs that search the web) that could lodge a brute force attack on the system by attempting multiple username password combinations until they are granted access. To stop this kind of attack on the system an intruder lockout could be used, whereby if a user exceeds a predefined number of incorrect login attempts their account is locked out and they can no longer access the system. This could be implemented by adding an additional field to the user table of the Postgres database for

user locks and either setting a default time for the user lock such as two hours, or requiring that the database administrator manually unlock the account.

A further method of preventing these robot attacks would be to generate a CAPTCHA image such as that in figure 22 below, where a code is printed slightly distorted against a background image. The idea of this capture image is to differentiate between a human and a machine user. While this code will be easily readable by any human user it will be rather difficult for a computer program without advanced text recognition algorithms. This however has its disadvantages in that it could become an inconvenience for users having to enter this code especially if they use the system regularly.

Figure 22: Example CAPTCHA image

The benefit of having this type of authentication is the ability to extend and customise the system to include different encryption algorithms and the ability to easily change the security policies associated with different user groups. All of this aids in the extensibility of the system which was one of the major design goals.

Another important consideration with this proposed authentication mechanism was the administration of user's accounts. This was an important consideration as one of the main design goals was that the system was easy to maintain.

The second authentication mechanism that was considered in this project was using the current School of Mathematics and Computer Sciences (MCS) Kerberos based security system to handle authentication into the database. This means that if a user attempts to perform a restricted operation then they will need to login via the MCS system. This is done using a simple PHP redirect to the MCS login page. When a user logs into the MCS system a PHP server variable is set with the username and group(s) that the user belongs to. These variables can then be used to implement authorisation at the PHP layer of the system.

Using this system however, means that any authorisation in the system will need to be either based on the already established user groups Students, Staff and NZSL or on a

per user basis. An alternative to this is to use a hybrid approach were the MCS system handles the user authentication and the user name is then checked against the database to determine the access group that this user falls into. This allows for the creation of new user groups in the Database system without requiring that these groups be set up in the MCS system.

Advantages of this mechanism are that all the administration of user's accounts and passwords is handled by either the MCS or SCS systems and thus the database interfaces need not support such functionality as updating passwords.

A Disadvantage of this system on the other hand is that for any user to have more than general access permissions that user must have either a Victoria University SCS account or an MCS account. This is not a large problem at present with the current system aimed at an internal audience consisting mainly of users from the DSRU and Victoria University sign language students. However into the future with the database being delivered to the wider community (subject to the DSRU obtaining the dictionary publishing rights) it may be advantageous for outside users such as the Deaf Association of New Zealand to have authorisation to update the database.

5.3 SQL Injection Attacks

A further security consideration to ensure the data integrity in any web based system is the detection and prevention of SQL Injection attacks. An SQL injection attack is the technique of "exploiting web applications that use client-supplied data in SQL queries without stripping potentially harmful characters first" (SPI Labs, 2002).

An example of the negative implications of such an attack can be illustrated by the following example. Assume the Dictionary was using the first authentication mechanism discussed in section 6.2 using a web based form and contained the following database table.

Field Name	Field Type	Description
User_ID	Integer	Primary key
Username	Varchar(10)	Username
Access_Group	Varchar(10)	Access permissions granted to this user
Password	Varchar(100)	One way encrypted hash of the password

Table 5-1 Hypothetical Database Table Dictionary_Users

Assuming the system created an SQL Database Query such as that illustrated in figure 23 using values entered by the user without any input validation it would be possible to enter ")' OR 1=1; --" into the password field in order to produce the query illustrated in figure 24.

```
SELECT Access_Group FROM Dictionary_Users WHERE Username =
'$UserName' AND Password = 'MD5($Password)';
```

Figure 23: Original Database Query

This would allow the user to gain unauthorised access to the database and thus the ability to modify the dictionary data, the database structure or even delete the whole database using UPDATE, ALTER or DROP SQL commands.

```
SELECT Access_Group FROM Dictionary_Users WHERE Username =
'$UserName' AND Password = 'MD5()' OR 1=1; --)';
```

Figure 24: Database Query after SQL Injection Attack

There are number of methods to detect and therefore prevent the effect of these attacks. Such methods include simply escaping any special characters found in input from the user. This would turn the Query displayed in figure 24 above into that shown in figure 25 below.

```
SELECT Access_Group FROM Dictionary_Users WHERE Username =
'$UserName' AND Password = 'MD5(\)\' OR 1=1\; \-\-)';
```

Figure 25: Database Query after SQL Injection Attack with input validation

When the above query is executed on the Database it will fail as there will not be a user who's password is the MD5 hash of "\)\' OR 1=1\; \-\-". This was the method that I chose to use in the implementation of the dictionary system. This was due to the fact that there are already PHP functions to perform these actions addslashes() and html_entities() which escape the appropriate characters for the database and html respectively.

Other methods that were considered included the use of regular expressions to detect any of these special characters at the PHP server layer of the system before these queries are submitted to the database. This method was decided against as it would require extra time to validate the input. Also several of the input fields in the system involve text that may contain these characters, which means that this method may produce a lot

of false positives. It was still possible however to perform some level of value checking at the web layer before sending queries to the Database.

A further method which was explored includes that introduced by Boyd and Keromytis 2004. This article presents a "practical protection mechanism against SQL injection attacks" which applies the "concept of instruction-set randomization to SQL, creating instances of the language that are unpredictable to the attacker" (Boyd and Keromytis, 2004).

This is achieved by appending an integer as a suffix to all of the SQL keywords as illustrated in table 6-2 and forwarding all Database Queries to an intermediate proxy between the CGI client and the DBMS.

Original Query	Modified Query
SELECT * FROM CITATION WHERE Citation_ID = 'computer';	SELECT123 * FROM123 CITATION WHERE123 Citation_ID = 'computer';

Table 5-2 Use of SQLrand in preventing SQL injection attacks

This proxy implements a modified SQL parser which expects all keywords to have this integer appended. This means that if the attackers' SQL injection contains any of the SQL keywords then this query will be rejected by the intermediate proxy as these keywords will not contain the random integer value.

However the main problem with implementing this approach in the dictionary is that at present this is not a common practice, and therefore could cause complications for future development of the system.

5.4 Maintenance

A major requirement identified early in the design of this system was to reduce the amount of maintenance that was required. To achieve this goal I have written a shell script that can be run as a Unix Cron job in the MCS system to vacuum and back up the database to the file system. This will help to reduce the amount of maintenance that is needed to be performed. Vacuuming of the database will carry out any necessary garbage collection from the Postgres Database in order to increase the performance of the overall system. This script will also create periodical back ups of the database to reduce the burden of restoring it from tape in the event that the data is corrupted.

6 Conclusions and Future Work

6.1 Conclusions

At the completion of this project, all of the major project objectives have been achieved. All relevant data has been extracted from the 4th Dimension Database and incorporated into a new full strength SQL based Postgres Database. The structure of this database is similar in many ways to the original 4th Dimension database with some major improvements in terms of database normalisation and added data and referential integrity constraints.

A prototype Dictionary User Interface has been designed, built and deployed on the MCS Apache web server allowing for the searching and updating of signs based on properties of the sign and English word equivalent.

The widespread adoption of the internet into society has seen the migration of numerous legacy systems onto the World Wide Web. This project highlights many of the benefits and some of the inherent difficulties involved with converting one such legacy system.

Some of these benefits include a more accessible system, with the ability to support multiple concurrent users. The speed at which users are able to search for and view information regarding NZSL signs has also been greatly increased. This is illustrated by the performance tests in section 5.6.4 where the average time required to search for a sign is now only 140ms compared with 99s in the 4th Dimension Database.

A lot of the difficulties encountered in this project were a result of a lack of available knowledge and information resources on the 4th Dimension Database. The effect of many of these problems could have been largely mitigated if documentation on the original system had been produced or comments had been included within the database modules.

A lot of time and effort in this project was also spent manually cleaning and comparing the data extracted from the old 4th Dimension database in order to manipulate it into a consistent state such that it could be inserted into the new database. Much of this work could have been avoided had the original database contained adequate data integrity constraints. However, as is common for legacy systems, this was not the case.

6.2 Future Work

6.2.1 User Acceptance Testing

The next important phase in this project will be the user acceptance testing with members of the DSRU. This phase will no doubt uncover further requirements that will need to be addressed before the end system is delivered.

6.2.2 Audit Trail

In order to protect the integrity of the data, it is vital that there be some mechanism set up to record all updates to the database so that these can be rolled back if necessary to prevent data from being lost or corrupted. At present any updates to the data in the database simply overwrite the previous data.

This is not such a critical problem at present with the database being hosted on the MCS system where regular tape backups are run overnight, but this is not an optimal solution for the long term.

6.2.3 Image Optimization

While converting the postscript images to PNG format has significantly reduced the file size of the sign images from approximately 100 kilobytes to approximately 20 kilobytes, these images could still be further optimized without reducing the quality of the images. This would give a performance improvement to the Dictionary especially on pages which contain a large number of these images.

6.2.4 Special Characters

For the Dictionary to be culturally sensitive and retain its credibility, it is essential that the special characters that had to be removed from the Maori and Foreign language words be restored. This will require these characters be manually checked against those in the printed dictionary.

6.2.5 Video Clips

A further requirement that will need to be addressed is the extension of the dictionary interface and Database to accommodate the new video footage and information that is presently being collected by the DSRU.

6.2.6 User Interface

To make this system more accessible and user friendly it would be beneficial to redesign or improve upon the prototype user interface with input from the DSRU. With input from the DSRU it may be possible to reduce the amount of information that has to be displayed regarding each sign. This could be achieved by only displaying the most significant information about each sign with the ability to drill down. This was not possible in the development of the prototype system as I was not in a position to determine the significance of this information.

6.3 Future Extensions

There are many ways in which this system could be further extended and improved upon.

6.3.1 Forum

It may be beneficial to have some sort of bulletin board or forum linked to the NZSL Dictionary. This forum could be used as a communication mechanism for researchers working on the NZSL Dictionary.

There are a number of Open Source PHP based forum packages that could be used for such a purpose. One such package is phpBB2. I have installed and created some simple forums using this package in order to demonstrate to the DSRU. This message board is illustrated in figure 26. Such a system however would require some maintenance and therefore could be considered as a future project for the DSRU.

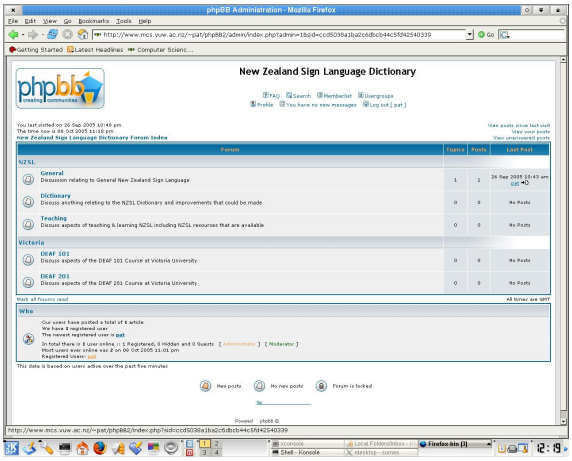


Figure 26: Forum for discussion of aspects of NZSL and general Deaf Culture

6.3.2 Virtual Human Signing

The use of a virtual human figurine to sign the NZSL signs included in the dictionary or as a virtual guide could also be a good extension of this project. This could be based on the work undertaken in the School of Computing Science at the University Of East Anglia in the ViSiCAST –Virtual Signing Capture Animation Storage and Transportation project— (Bangham et al, 2000) and the e-Sign project undertaken at the University of Hamburg (Hanke, Popescu and Schmaling, 2003). These projects have set out to transform the HamNoSys encoding of British Sign Language, "*Deutsche Gebärdensprache*" German Sign Language and "*Nederlandse Gebarentaal*" Dutch Sign Language signs into SiGML (Sign Gesture Mark-up Language). SiGML is an XML based specification which describes aspects of signs such as the hand shape(s), movement and location in signing space as well as information regarding the signers facial expressions. Using this SiGML, animation cues can be sent to a virtual avatar to create animation frames. Figure 18 shows the avatar "Visia" from the ViSiCAST project signing a television news item.

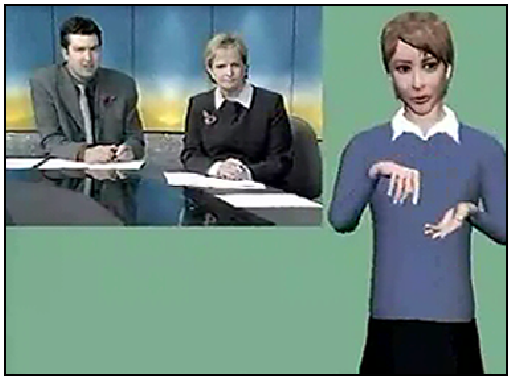


Figure 27: Virtual Signer avatar "Visia 2" signing a television news item

The ViSiCAST project has created a prototype java program for the transformation of HamNoSys into SiGML. This program is based on Terence Par's Antlar translator generation system. The output of running this program on the HamNoSys encoding for the DGS sign "Going-To" (figure 28) is shown in figure 29.

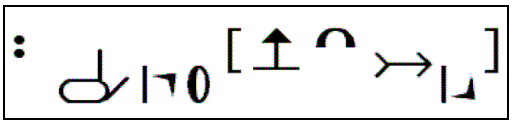


Figure 28: HamNoSys representation of the German Sign Language sign "Going-To"

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE sigml SYSTEM "sigmlv0.dtd">
<sigml>
<avatar url="Simon.ava" id="A" alt="Simon"/>
<sign gloss="GOING-TO">
      <hamnosys_sign lr_symm="parallel">
            <handposture
                  handshapeclass="ham_finger2"
                  thumbpos = "ham_thumb_outmod"
                  extfidir="direction_uo"
                  palmor="direction_l">
            </handposture>
      <par_movement>
            <straightmovement
                  direction="direction_o"
                  curve="direction_u"
            />
            <handposture extfidir="direction_do"/>
      </par_movement>
      </hamnosys_sign>
</sign>
</sigml>
```

Figure 29: SiGML representation of the German Sign Language sign "Going-To"

The use of such a virtual signer might provide Deaf users with easier access to the dictionary information as it would be presented in their natural language.

6.3.3 Movement Specification

A future extension of this project would be the ability to search for signs based on a specification of the movement involved. This could be based on the manipulation of a three dimensional human figurine similar to the virtual signing avatars discussed in section 7.2.3. This would allow novice signers the ability to look up signs they had encountered but not understood in order to ascertain their meaning. A 3D based virtual signer such as this, with the ability to specify signing movement was also investigated by Scott (2003).

6.3.4 Mobile and PDA Access

A further extension of this could be to redesign the prototype user interface for smaller display mobile devices such as mobile phones and PDA's. This would further improve the accessibility of information regarding NZSL.

Bibliography

4th DIMENSION. URL: http://www.4D.com/. Accessed October 2005

ACI. 4th Dimension Language – Reference Manual. 1999

APTE, V., HANSEN, T., REESER, P. Performance Comparison of Dynamic Web Platforms. 2001

BANGHAM, JA., COX, SJ., ELLIOT, R., GLAUERT, JRW., MARSHALL, I. Virtual Signing: Capture, Animation, Storage and Transmission – an Overview of the ViSiCAST Project. 2000

BANGHAM, JA., COX, SJ., LINCOLN, M., MARSHALL, I. Signing for The Deaf Using Virtual Humans. 2000

BOYD, S., AND KEROMYTIS, A. SQLrand: Preventing SQL injection attacks, 2004.

CLARK, D. PHP Security Mistakes. 2004 URL: http://www.devshed.com/c/a/PHP/PHP-Security-Mistakes/ Accessed October 2005

DEAF ASSOCIATION OF NEW ZEALAND. URL: http://www.deaf.co.nz/. Accessed October 2005.

DEAF TODAY. Various Archived Newspaper Articles. URL: http://www.deaftoday.com/. Accessed October 2005

DYSON, R. New Zealand Sign Language Bill First Reading. 22 June 2004

HANKE, T. POPESCU, H., SCHMALING, C. eSIGN – HPSG-assisted Sign Language Composition. Gesture Workshop 2003

HANKE, T., SCHMALING, C. Hamburg Sign Language Notation System. University of Hamburg. URL: http://www.sign-lang.uni-hamburg.de/Projects/HamNoSys.html. Accessed October 2005

JEPSON, B., PostgreSQL vs. MySQL: Building Better Databases. 2001. URL: http://www.webtechniques.com/archives/2001/09/jepson/. Accessed October 2005

JOYE. P., JANSEN, M., CASTAGNETTO, J., KNOWLES, A., COX, T., PARISE, J., BAKKEN, S. The PEAR Group. PHP Extension and Application Repository. URL: http://pear.php.net/. Accessed October 2005

KENNEDY, G., Ed. A Dictionary of New Zealand Sign Language. Auckland University Press/Bridget Williams Books, 1997.

LOCKER MCKEE, R., MCKEE, D., New Zealand Sign Language Grammar : A guide for learners. Wellington [N.Z.] : School of Linguistics and Applied Language Studies, Victoria University of Wellington, 2002.

MAHONY, J. Sign Language Gets Own Easy to Use Dictionary. New Zealand Herald. 15-2-2003.

MOOKHEY, K., BURGHATE, N. Detection of SQL Injection and Cross Site Scripting Attacks. 2004

MYSQL AB. URL: http://www.mysql.com/. Accessed October 2005

New Zealand Sign Language Bill, 2005 (House of Representatives)

PENMAN, P. Deaf way, deaf view: a study of deaf culture from a deaf perspective. Master's thesis, Victoria University, 1999.

PERUDE , T. PHPBuilder.com - MySQL and PostgreSQL Compared, 2000. URL: http://phpbuilder.com/columns/tim20000705.php3?page=1. Accessed June 2005.

PLUGGERS SOFTWARE. PostgreSQL PLUGIN. URL: http://www.pluggers.nl/postgresql_plugin.html. Accessed October 2005

SAETHER BAKKEN, S., AULBACH, A., SCHMID, E., WINSTEAD, J., TORBEN WILSON, L., LERDORF, R., ZMIEVSKI, A., AHTO, J. PHP Manual. The PHP Documentation Group, 2002

SCOTT, S. A Search Facility for a New Zealand Sign Language Dictionary. 2003

SPI LABS. SQL Injection : Are Your Web Applications Vulnerable? White Paper. 2002

STRANGE, H. Office of the Clerk of the House of Representatives. URL: http://www.clerk.parliament.govt.nz/Programme/Committees/Submissions/jenzsl.htm/ Accessed October 2005

THE POSTGRESQL GLOBAL DEVELOPMENT GROUP. PostgreSQL 8.0.3 Documentation.

TITCHKOSKY, L., ARLITT, M., AND WILLIAMSON, C. A performance comparison of dynamic web technologies. SIGMETRICS Perform. Eval. Rev. 31, 3 (2003), 2-11.

VICTORIA UNIVERSITY. Deaf Studies 101. URL: http://www.vuw.ac.nz/lals/courses/DEAF-101.aspx Accessed October 2005

Appendix A SQL Database Schema Definition

--Stores information about NZSL signs

```
CREATE TABLE Citation(
    Gloss varchar(40),
    VariantNumber int,
    Example varchar(55),
    Receipe Text,
    Hint Text,
    Comments Text,
    Signer varChar(40) CONSTRAINT CitationRefPeople REFERENCES
    People(FullName),
    Tape_ID varChar(20) CONSTRAINT CitationRefTapes REFERENCES Tapes,
    Sequence int,
    WN_Valid varchar(3),
    WN_CVS varchar(40),
    CH_Valid varchar(3),
    CH_CVS varchar(40),
    AK_Valid varchar(3),
    AK_CVS varchar(40),
    Waikato_Valid varchar(3),
    HawkesBay_Valid varchar(3),
    Taranaki_Valid varchar(3),
    Manawatu_Valid varchar(3),
    Otago_Valid varchar(3),
    Southland_Valid varchar(3),
    HamNoSys varchar(80),
    Directional Boolean,
    Orienting Boolean,
    Variable_Tab Boolean,
    One_Or_Two_Hand Boolean,
    Say varchar(40),
    Repeats int,
    Locatable Boolean,
    Complex varchar(20),
    Head varchar(20),
    Eyebrows varchar(20),
    Eyes varchar(20),
    Mouth varchar(30),
    Tonuge varchar(20),
    Cheeks varchar(20),
    Shoulders varchar(20),
    Body varchar(30),
    Elbows varchar(20),
    Feet varchar(20),
    Last_Updated Date,
    Citation_ID varchar(45) CONSTRAINT CitationPK PRIMARY KEY,
    Batch_Number int,
    QA Boolean,
```

```
    Fingerspelling Boolean,
    Origin varchar(3),
    Created Date,
    QA_Text varchar(6),
    QA_Artwork varchar(6),
    QA_Notation varchar(6),
    QA_Synthesis varchar(6),
    QA_Performance varchar(6),
    QA_Deaf_Review varchar(6),
    QA_Final_Review varchar(6),
    Lip_Pattern varchar(20),
    Number_Incorp Boolean,
    Special_Valid varchar(3),
    Special_CVS varchar(40),
    HNS_HS varchar(15),
    HNS_TAB varchar(20),
    Tab_ADJ int,
    HS_class int,
    HNS_HNS_sort varchar(80),
    Concise Boolean
);
```

--All words includes both synonyms and Headwords

```
CREATE TABLE Word(
    Word varchar(50) PRIMARY KEY
)
```

--Description of how a sign is used

```
CREATE TABLE Usage(
    Usage varchar(20) CONSTRAINT UsagePK PRIMARY KEY,
);
```

--Information regarding the VHS tapes used to record signer information sessions

```
CREATE TABLE Tapes(
    Tape_ID varchar(20) CONSTRAINT TapesPK PRIMARY KEY,
    Last_Backed_Up Date,
    Title varchar(20),
    Length int,
    Where_Held varchar(20),
    Prefix varchar(10),
    Number int
);
```

--Information regarding a synonym of an NZSL sign meaning

```
CREATE TABLE Synonyms(
    MeaningID int,       --Link to the meaning this synonym belongs to
    Synonym varchar(40), --The synonym itself
    Rank int,            --The rank of this synonym in the meaning
    CONSTRAINT SynonymsPK PRIMARY KEY(MeaningID,Synonym),
    CONSTRAINT RankUnique UNIQUE(MeaningID,Rank)
);
```

--Information regarding a signer involved in the information gathering process

```
CREATE TABLE People (
    FullName varchar(40) CONSTRAINT PeoplePK PRIMARY KEY,
    Surname varchar(20),
    FirstNames varchar(20),
    Address text,
    Phone1 varchar(20),
    Phone2 varchar(20),
    Phone3 varchar(20),
    AgeGroup varchar(20),
    Region varchar(20),
    Notes text
);
```

--Information regarding words that were identified as important for investigation by linguists

```
CREATE TABLE Merged_Word_List(
    Ahlgren boolean,
    CofE boolean,
    Costello boolean,
    Flynn boolean,
    Frisby boolean,
    Johnston boolean,
    Levitt boolean,
    Nation boolean,
    Sternberg boolean,
    Word varchar(45),
    Authors varchar(27)
);
```

--The relation between a words meaning and its citation/sign

```
CREATE TABLE Meaning_Citations(
    CitationID varchar(45) CONSTRAINT MeaningCitationsRef REFERENCES Citation,
              --The citation that represents this meaning
    HeadWord varchar(50),  --The headword of the Meaning that this citation
    relates to
    MeaningRank int, --The rank of the Meaning that this Citation relates to
```

```
    CONSTRAINT Meaning_CitationsPK PRIMARY
    KEY(CitationID,HeadWord,MeaningRank)
);
```

--The meaning of a sign or English word

```
CREATE TABLE Meaning(
    Headword varchar(50) REFERENCES Word(Word),
    MeaningRank int,
    Use text,
    Comments Text,
    CONSTRAINT MeaningPK PRIMARY KEY(Headword,MeaningRank)
);

CREATE TABLE Keyword(
    Keyword varchar(45) CONSTRAINT KeyWordPK PRIMARY KEY,
);
```

--Relationship between a word and it's meaning

```
CREATE TABLE Has_Meaning(
    Synonym varchar(50) REFERENCES Word,
    SynonymRank int,
    HeadWord varchar(50),
    MeaningRank int,
    CONSTRAINT Has_MeaningPK PRIMARY KEY(Synonym,Headword,MeaningRank),
    CONSTRAINT Has_Meaning_Unique UNIQUE(SynonymRank,HeadWord,MeaningRank),
    CONSTRAINT Has_MeaningReferencesMeaning FOREIGN KEY (HeadWord,MeaningRank)
    REFERENCES Meaning (HeadWord,MeaningRank)
);

CREATE TABLE Citation_Keyword_List(
    Citation_ID varchar(45) CONSTRAINT CKLRefCitation REFERENCES Citation ON
    DELETE CASCADE,
    Keyword varchar(45) CONSTRAINT CKLRefKeyword REFERENCES Keyword ON DELETE
    CASCADE,
    CONSTRAINT Citation_Keyword_ListPK PRIMARY KEY(Citation_ID, Keyword)
);

CREATE TABLE Citation_Usage_List(
    Citation_ID varchar(45) CONSTRAINT CULRefCitation REFERENCES Citation ON
    DELETE CASCADE,
    Usage varchar(20) CONSTRAINT CULRefUsage REFERENCES Usage ON DELETE
    CASCADE,
    CONSTRAINT Citation_Usage_ListPK PRIMARY KEY(Citation_ID, Usage)
);

CREATE TABLE SISSigners(
    Name varchar(40) CONSTRAINT SISSignersRefPeople REFERENCES
    People[FullName], --Problem with this line
    SIS_ID varchar(45) CONSTRAINT SISSignersRefSIS REFERENCES SIS,
```

```
    CONSTRAINT SISSignersPK PRIMARY KEY (Name, SIS_ID)
);

CREATE TABLE SISTapeIDs_Used(
    Tape_ID varchar(20) CONSTRAINT SISTapeIDs_UsedRefTapes REFERENCES Tapes ON
    DELETE CASCADE,
    Index int,
    SIS_ID varchar(45) CONSTRAINT SISTapeIDs_UsedRefSIS REFERENCES SIS ON
    DELETE CASCADE,
    CONSTRAINT SISTapeIDs_UsedPK PRIMARY KEY (Tape_ID, SIS_ID)
);

CREATE TABLE SIS_Gloss_List(
    Gloss varchar(40),
    SIS_ID varchar(45), CONSTRAINT SISRef REFERENCES SIS,
    Tape_ID varchar(20) CONSTRAINT TapeIDRef REFERENCES Tapes,
    Counter Time
);

CREATE TABLE SIS(
    SISDate Date,        --Renamed as Date may cause problems
    SISTime Time,        --Renamed as Time may cause problems
    Region varchar(20),
    TopicsCovered Text,
    Location varchar(20),
    Convenor varchar(20),
    Notes Text,
    SIS_ID varchar(45) CONSTRAINT SISPK PRIMARY KEY
);

CREATE TABLE Word_Counts(
    Word varchar(40),
    Count int,
    Constraint Word_CountPK PRIMARY KEY(Word,Count)
);

CREATE TABLE CVSTapeIDs_Used(
    Tape_ID varchar(20) CONSTRAINT CVSTapeIDs_UsedRefTapes REFERENCES Tapes ON
    DELETE CASCADE,
    Index int,
    CVS_ID varchar(40) CONSTRAINT CVSTapeIDs_UsedRefCVS REFERENCES CVS ON
    DELETE CASCADE,
    CONSTRAINT CVSTapeIDs_UsedPK PRIMARY KEY (Tape_ID, CVS_ID)
);

CREATE TABLE CVSSigners(
    Name varchar(40) CONSTRAINT CVSSignersRefPeople REFERENCES People
    [FullName],
    CVS_ID varchar(40) CONSTRAINT CVSSignersRefCVS REFERENCES CVS,
    CONSTRAINT CVSSignersPK PRIMARY KEY (Name, CVS_ID)
);
```

Appendix B 4D Database Structure