ReCALL **22**(2): 152–171. 2010 © European Association for Computer Assisted Language Learning
doi:10.1017/S0958344010000054

Sintéiseoir 1.0: a multidialectical TTS application for Irish

MÍCHEÁL MAC LOCHLAINN
Acadamh na hOllscolaíochta Gaeilge, Ollscoil na hÉireann Gaillimh, Roisín na Mainiach, Carna, Cúndae na Gaillimhe, Éire
(email: gaoluinn@eircom.net)

Abstract

This paper details the development of a multidialectical text-to-speech (TTS) application, *Sintéiseoir*, for the Irish language. This work is being carried out in the context of Irish as a lesser-used language, where learners and other L2 speakers have limited direct exposure to L1 speakers and speech communities, and where native sound systems and vocabularies can be seen to be receding even among L1 speakers – particularly the young.

Sintéiseoir essentially implements the diphone concatenation model, albeit augmented to include phones, half-phones and, potentially, other phonic units. It is based on a platform-independent framework comprising a user interface, a set of dialect-specific tokenisation engines, a concatenation engine and a playback device.

The tokenisation strategy is entirely rule-based and does not refer to dictionary look-ups. Provision has been made for prosodic processing in the framework but has not yet been implemented. Concatenation units are stored in the form of WAV files on the local file system.

Sintéiseoir's user interface (UI) provides a text field that allows the user to submit a grapheme string for synthesis and a prompt to select a dialect. It also filters input to reject graphotactically invalid strings, restrict input to alphabetic and certain punctuation marks found in Irish orthography, and ensure that a dialect has, indeed, been selected.

The UI forwards the filtered grapheme string to the appropriate tokenisation engine. This searches for specified substrings and maps them to corresponding tokens that themselves correspond to concatenation units.

The resultant token string is then forwarded to the concatenation engine, which retrieves the relevant concatenation units, extracts their audio data and combines them in a new unit. This is then forwarded to the playback device.

The terms of reference for the initial development of *Sintéiseoir* specified that it should be capable of uttering, individually, the 99 most common Irish lemmata in the dialects of An Spidéal, Músgraí Uí Fhloínn and Gort a' Choirce, which are internally consistent dialects within the Connacht, Munster and Ulster regions, respectively, of the dialect continuum. Audio assets to satisfy this requirement have already been prepared, and have been found to produce reasonably accurate output. The tokenisation engine is, however, capable of processing a wider range of input strings and when required concatenation units are found to be unavailable, returns a report via the user interface.

Keywords: Irish, TTS, speech synthesis, dialect

1 Background

Sintéiseoir 1.0 is an Irish language text-to-speech application which is in the early stages of development. It was created as part of the author's M.Sc., thesis (Mac Lochlainn, 2008), in which a multidialectical framework for Irish language TTS synthesis was designed. This research was carried out in the context of Irish as a lesser-used language, where learners and other L2 speakers have limited direct exposure to L1 speakers and speech communities, and where native sound systems and vocabularies can be seen to be receding even among L1 speakers – particularly the young. The intention is to develop a CALL resource to address these problems.

The research explores similar ground to that done in Trinity College, Dublin, and which resulted in the, currently mono-dialectical, speech synthesiser abair.ie (Coláiste na Tríonóide, 2008).

1.1 Terms of reference

The project was approached with regard to the academic level at which the work was being done and with regard to pragmatic constraints also: the research was carried out single-handedly and on a part-time basis. In light of these, the following terms of reference were set out:

- Develop a modular logical framework for Irish language TTS synthesis to support an unlimited number of dialects.
  - Textual input will not be restricted to current standard Irish spelling; thus dialectical written forms (historically established ones in particular) are valid, as indeed are all utterable strings.
  - Audio output will conform to specific dialectical forms as opposed to generalised regional abstractions.
- Develop a functioning software application based on the framework.
- The first version of the software will support the dialects of An Spidéal, Músgraí Uí Fhloínn and Gort a' Choirce, which are internally consistent dialects within the Connacht, Munster and Ulster regions, respectively, of the dialect continuum.
  - In view of time and resource limitations, however, support will be restricted, in the first version of the software, to forms permitted in *an lárchanúint*, which is a simplified, artificial dialect of the language.
- The first version of the software will only be required to successfully utter single words, but must be able to do so with equal facility in each of the three selected dialects.
- The software's initial vocabulary will be the ninety-nine most common lemmata of the Irish language, as identified in Buntús Gaeilge (1966).
- User input containing any of the 26 graphemes of the English alphabet and any of the the vowels "á", "é", "í", "ó" and "ú" marked diacritically with an acute accent will be considered valid.
- Prosodic processing will not be investigated nor implemented in the current research.
- Accurate, consistent grapheme-to-phone translation will be effected but no formal success/failure criteria will be defined with regard to intelligibility.

https://doi.org/10.1017/S0958344010000054 Published online by Cambridge University Press

2 Synthesis model

It was decided to base the framework on the concatenative synthesis model, in which samples of prerecorded human speech are joined together to produce phonic continua. The samples themselves may be phones (discrete units of sound, produced by the human articulatory apparatus), half-phones, diphones, triphones or larger units such as half-syllables, syllables, words, phrases and even whole sentences.

The current version of *Sintéiseoir* essentially implements diphone synthesis, albeit slightly augmented to deploy single phones to support monophonic utterances and half-phones to initiate and terminate polyphonic ones. In this concatenation model, discrete recordings are prepared of (ideally) all the phonotactically valid phone-to-phone transitions in the language to be synthesised; thus supporting all possible articulatory transitions.

3 The framework

3.1 The modules

The framework (Figure 1), which structures the dataflow of the TTS process, was developed before any program coding was done.

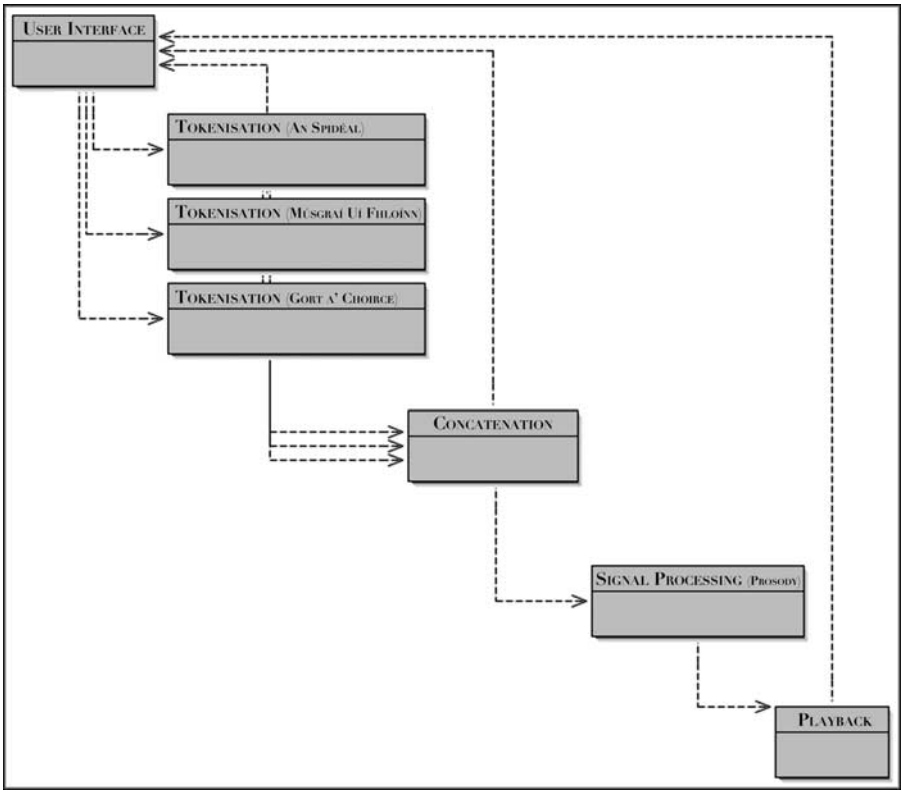

Fig. 1. The framework and dataflow

https://doi.org/10.1017/S0958344010000054 Published online by Cambridge University Press

The first module is a user interface which allows the user to submit a grapheme string for synthesis and select a dialect. This module also returns feedback such as error messages, generated by itself or by subsequent modules. It forwards the user input as a grapheme string in plaintext.

Following the UI are a set of tokenisation engines, arranged 'in parallel'. Figure 1 depicts the three dialects selected for this research but any number of additional dialects may be added at this stage. The tokenisation engine for the selected dialect accepts the grapheme string forwarded by the UI, searches for specified substrings and maps them, subject to the phonological system of that dialect, to corresponding tokens that themselves correspond to concatenation units.

The third stage in the data flow is the concatenation engine. This retrieves the relevant concatenation units from the appropriate file system. Their audio data are extracted, concatenated and inserted into a new audio data unit, the nature of which depends on the software environment in which the framework is implemented.

The fourth stage is the prosodic processing module. Although the terms of reference placed prosody outside the scope of the project it is an inescapable aspect of speech synthesis and will be addressed in future research, hence the inclusion of the module in the framework. For the time being, however, it is a functionless component which simply forwards the binary audio data unaltered.

The final module is the playback device.

3.2 Dictionary lookups

Due to the condition, stated in the terms of reference, that the framework should not be tied to current standard Irish spelling, it was decided to exclude dictionary lookups in favour of a purely rule-based tokenisation methodology. This is not set in stone however; such functionality may be added to the framework in the future, in which case the requisite module would be placed between the user interface and the tokenisation engines. It would almost certainly refer to a collection of grapheme and token string pairs stored in a database and the results of successful lookups would be forwarded directly to the concatenation engine.

4 Tokenisation methodology

Tokenisation is, in this context, the process of replacing a grapheme string with a string of textual hints or indicators, called tokens, that refer to corresponding phones; thus a token string is a textual representation of a phone sequence to be uttered. The principle is essentially that of traditional phonetic or, indeed, phonemic transcription.

There are 18 graphemes in the Irish alphabet (Figure 2). The non-native (that is, English) graphemes "j" and "v" have some small historical presence in loan words. For reasons that will be discussed presently, justification exists for permitting these, and also the English graphemes "k", "q", "w", "x", "y" and "z", in the framework, if only to support scientific notation and a small number of uncommon gaelicisations.

The system of broad transcription traditionally used by scholars of Irish linguistics was considered an ideal notation for use in the tokenisation engines. This system, which was based on that of Professor Daniel Jones in 'An Outline of English Phonetics'

Short vowels:	/a/ /e/ /i/ /o/ /u/
Long vowels:	/a:/ /e:/ /i:/ /o:/ /u:/
The neutral vowel:	/ə/
Dipthongs:	/ai/ /au/ /uə/ /iə/
Velarised consonants:	/b/ /k/ /x/ /kw/ /d/ /ɣ/ /f/ /g/ /h/ /l/ /m/ /n/ /ŋ/ /ŋg/ /p/ /r/ /s/ /t/ /v/ /w/ /z/
Palatalised consonants:	/b′/ /k′/ /x′/ /d′/ /d′z′/ /ɣ′/ /f′/ /g′/ /l′/ /m′/ /n′/ /ŋ′/ /ŋ′g′/ /p′/ /r′/ /s′/ /t′/ /v′/ /ek′s/ /z′/
Native graphemes:	a b c d e f g h i l m n o p r s t u
Non-native graphemes:	j k q v w x y z

Fig. 2. The phones and graphemes of Irish, as permitted in an lárchanúint

(Ó Cuív, 1944), and which differs somewhat from that of the international phonetic alphabet (IPA), has been subject to minor modification by various authors since its initial use but has remained generally consistent. It follows the common form of a pair of right-leaning stroke characters enclosing one or more alphabetical ones. Long vowels are indicated by a following colon and palatalisation of consonants by a following prime. The system is also shown in Figure 2, which lists the full inventory of the 56 phones permitted in *an lárchanúint* (Ó Baoill, 1986a; Ó Baoill, 1986b).

4.1 Filenames

Having decided upon the tokenisation format, it was clear that the same notation was appropriate as a file-naming convention for the sound files carrying the concatenation units, albeit with the bounding strokes omitted as these are reserved characters in filepath notation. Thus, the file b.wav carries the phone /b/, the file ŋ′g′.wav carries the phone /ŋ′g′/, the file b′e.wav carries the diphone /b′e/ and so on. Initial and terminal half-phones are denoted by prefixing the characters 0 and 1 respectively to the filenames: 0b.wav, 1b.wav and so on.

5 The phone inventory

The 56 phone inventory of *an lárchanúint* was slightly augmented following aspects of the broad transcription used in Ó Cuív (1944), Ó Duilearga (1981) and Ó Sé (2000) as it was felt that emphasis on simplification in *an lárchanúint* had resulted in certain significantly different phones being represented by the same notational characters:

- /a/, which denotes both Cardinal 4, /a/, and Cardinal 5, /ɑ/.
- /ə/, which denotes both the neutral vowel, /ə/, and the near-close near-front unrounded vowel, /ɪ/.

https://doi.org/10.1017/S0958344010000054 Published online by Cambridge University Press

These differences may be illustrated by contrasting /bʹan/ (“woman”, in the nominative singular) with /bɑn/ (“women”, in the genitive plural), and /ɑ:skə/ (instance) with /isʹkʹɪ/ (water). Despite the implicative nature of broad transcription, and the assertion that “This minor variation [in the pronunciation of the neutral vowel] (is not significant and) does not change the word” (Ó Baoill 1986a:3), these distinctions *are* significant, in that they indicate perceptible velarisation and palatalisation, and they *do* change words, inasmuch as inappropriate use is perceived as highly unnatural by L1 speakers. To remedy this, it was decided to admit the phones /ɑ/, /ɑ:/, and /ɪ/ to the inventory.

At the same point during the research, it was decided to admit the phone /wai/ into the inventory also. The phone /ekʹs/ had been permitted in *an lárchanúint* to cater for the Irish renderings of a very small number of scientific and mathematical terms such as “X-ray”, “X-chromosome” and “X-axis”. /wai/ (“Y-chromosome”, “Y-axis”) had not been included, however, and it seemed reasonable to complete the complementary pair here. (On subsequent reflection, it was realised that /ʏe:/ might be an equally valid alternative to /wai/, as it conforms to traditional native pronunciation of this non-native letter. This question will be revisited during future work.)

6 Designing the diphone set

Following Lenzo and Black (2000), it was decided to prepare a diphone inventory of every possible Irish phone-to-phone transition by starting with a list of every 2-permutation of the permitted phones and to then reduce this by excluding phonotactically invalid clusters. Since the following phones are only there to support a tiny number of loan words, it was decided that they should be admitted as discrete phones but excluded as diphone components:

- **/ek**ʹ**s/** and /**wai/:** Used as prefixes or suffixes only. A small number of diphones may be compiled and recorded in the future to accommodate common usage.
- **/w/:** Required for only one uncommon loan-word (wig-wam) given in Ó Baoill (1986b). This word is not given at all in Ó Domhnaill (1977), which is the most comprehensive Irish-English dictionary based on current standard orthography.
- **/kw/:** Required for only one uncommon loan-word (for the English ‘quinine’) in Ó Baoill (1986b), and one other *very* uncommon loan-word (for ‘quinol’) in Ó Domhnaill (1977).

The total number of diphones is equal to the total number of 2-permutations (r) of the remaining 56 phones (n), which was found using the formula:

$$P(n, r) = (n!)/(n-r)!$$

$$\therefore$$

$$P(56, 2) = (56!)/(56-2)!$$

$$= 3,080$$

The online facility provided in Schmitt (2009) was used to generate a full list of 3,080 2-permutations of the ordinal numbers 1 to 56. This list was then subjected to a find

https://doi.org/10.1017/S0958344010000054 Published online by Cambridge University Press

and replace procedure to substitute each of the 56 phone symbols for corresponding numerics ('a' for each '1', 'a:' for each '2' and so on), resulting in a list of phone pairs. This was then edited manually to remove phone transitions considered invalid according to the following criteria:

- Consonant clusters other than those permitted in Ó Baoill (1986a) or Ó Baoill (1986b).
  - It was originally intended to excise all such clusters but close inspection forced a reassessment. *An lárchanúint* is an extremely prescriptive paradigm and certain clusters such as /l′s′/ and /g′s′/ were not listed as permitted. However, these clusters occur in common words such as *ailse*, *soilse*, *éigse* and *laigse*, which are accorded standard status in Ó Domhnaill (1977) and which cannot be uttered without them, even within the strictures of *an lárchanúint*. This suggested some oversight during editing or typesetting and these clusters were, therefore, retained.
- All short-vowel clusters, as these would equate to diphthongs, of which the only four that are permitted in *an lárchanúint* exist as phones already.
- All diphthong clusters, as these would equate to quodrothongs, which are nonexistent in Irish.
- All vowel and diphthong clusters, as these would equate to triphthongs, which are nonexistent or extremely rare in Irish, or indicate glides, which are not generally within the scope of broad transcription.
- Unutterable clusters. This is a slightly subjective criterion, and no formal verification was attempted due to time constraints. However, such clusters are, by their nature, fairly obvious to any L1 or competent L2 speaker.
- Clusters considered unnatural to the language. This was, of course, a considerably subjective criterion, albeit informed by linguistic competence. It mostly affected clusters containing phones such as /z/ and /d′z′/, which are generally found only in loan words, occurring natively only in certain dialects, but also applied to clusters that are utterable only with difficulty by L1 speakers, or for which examples could not be found in either standard or dialectical vocabularies.

Most long-vowel clusters were retained as a significant proportion (ten out of a total thirty permutations) were found in words given in Ó Domhnaill (1977). Of the other twenty, only the /a: ɑ:/ and /ɑ: a:/ transitions were considered sufficiently unnatural to warrant exclusion.

7 Designing the carrier material

Suitable carrier material (that is, uttered sound from which the concatenation units are to be extracted) was required to facilitate the recording of the diphones, phones and half-phones. Following Lenzo and Black (2000), it was decided to embed these in specially designed nonsense words:

> We believe that the use of nonsense carrier material makes the delivery of the diphones more consistent. Also, the pronunciation of a phonetic string is more clearly defined in these nonsense words than in elicited natural words. We generate carrier phrases so that, where possible, we can extract the diphones

https://doi.org/10.1017/S0958344010000054 Published online by Cambridge University Press

> from the middle of a word. As it takes time for the human articulation system to start, we do not want to extract diphones from syllables at the start or end of words, unless these transitions to or into silence (SIL) are part of the diphone in question.
>
> Lenzo and Black (2000)

7.1 Diphones

Following this approach, a set of single phones, compatible within the sound system of the language, was prefixed and appended prosthetically to each item in the final diphone set, with the result of creating a list of nonsense words, designed to be easily and intuitively utterable to L1 speakers, and from which the diphone set could be extracted after recording:

- The vowel /ə/ was deployed next to velarised consonants.
- The vowel /ɪ/ was deployed next to palatalised consonants.
- The consonant /bʹ/ was deployed next to front vowels.
- The consonant /b/ was deployed next to back vowels.

For example:

- /ə/-/bc/-/ə/
- /ɪ/-/bʹcʹ/-/ɪ/
- /bʹ/-/esʹ/-/ɪ/
- /ɪ/-/sʹe/-/bʹ/
- /ə/-/sɑ/-/b/
- /b/-/ɑs/-/ə/

7.2 Single phones

Single phones were simply uttered and recorded in their entirety.

7.3 Half-phones

Compiling the half-phone inventory was slightly more involved. As these are only used in the framework to initiate and terminate phonic continua they, paired with their complementary halves, invariably precede or follow dissimilar phones, with the resultant influence on their articulation. Merely chopping single phones in half worked well enough for initial half-phones but produced audibly unnatural results at the terminations. The following strategy, in which the initial and terminal half-phones of the sequences are captured, was devised to remedy this:

- The pattern <consonant> - /ɪ/ - <consonant> was deployed to capture palatalised consonants.
- The pattern <consonant> - /ə/ - <consonant> was deployed to capture velarised consonants.
- The pattern <vowel> - /bʹ/ - <vowel> was deployed to capture front vowels.
- The pattern <vowel> - /b/ - <vowel> was deployed to capture back vowels.

https://doi.org/10.1017/S0958344010000054 Published online by Cambridge University Press

For example:

- /c/-/ə/-/c/
- /c′/-/ɪ/-/c′/
- /e/-/b′/-/e/
- /ɑ/-/b/-/ɑ/

8 Recording the carrier material

Bearing in mind the terms of reference (and particularly limitations of time and resources), only the diphones necessary to facilitate synthesis of the 99 lemmata given in Buntús Gaeilge were recorded and prepared as concatenation units. All the permitted half-phones and single phones were recorded, however.

Access to a dedicated recording environment was not available, therefore recording was done in a room in an ordinary dwelling house in a remote, therefore quiet, area. As many surfaces as possible, particularly those in front of and behind the speakers, were covered with heavy quilts to attenuate reverberation. The microphone was a Rode NT1-A, placed within an SE Electronics Reflexion Filter (to further reduce reverberation) and behind a 4 inch pop screen. It was connected to an Apple Mac computer via an M-Audio Firewire Audiophile sound card. The recording software was Audacity version 1.2.5.

8.1 Speakers

L1 speakers from An Spidéal and Gort a' Choirce were selected to provide voices for these dialects. Both are extremely fluent and knowlegeable native speakers, and are also formal, experienced scholars of Irish linguistics. The author himself provided the voice for the dialect of Músgraí Uí Fhloínn. The dialects were selected partly because suitable speakers were available but also because they provide a broad spectrum of geographical and linguistic representation (Figure 3).

Speakers were provided with a list of sounds to utter, with the carrier material typewritten in the broad phonetic transcription described previously. They were asked to speak at as constant a volume and pitch as possible, and with no prosody whatsoever.

The author did not have any significant problem with these requirements but that was undoubtedly because, *as* the author, he understood profoundly what was required and *not* because of any superior linguistic ability; as noted in Lenzo and Black (2000), delivering diphones is not a particularly natural endeavour, after all.

After a number of pilot sessions the other speakers were able to regulate volume, although some prompting was required as they reached the end of the list, as they tended to become quieter. However, given their scholastic background as linguists, prosody proved to be surprisingly hard to eradicate and several re-takes were required – even after work in the pilot sessions indicated that they had it under control. Interestingly, in both cases the instruction "*tabhair amach ar nós róbait é*" ("speak it like a robot") resulted in immediate improvement, whereas previous requests had been phrased more formally.

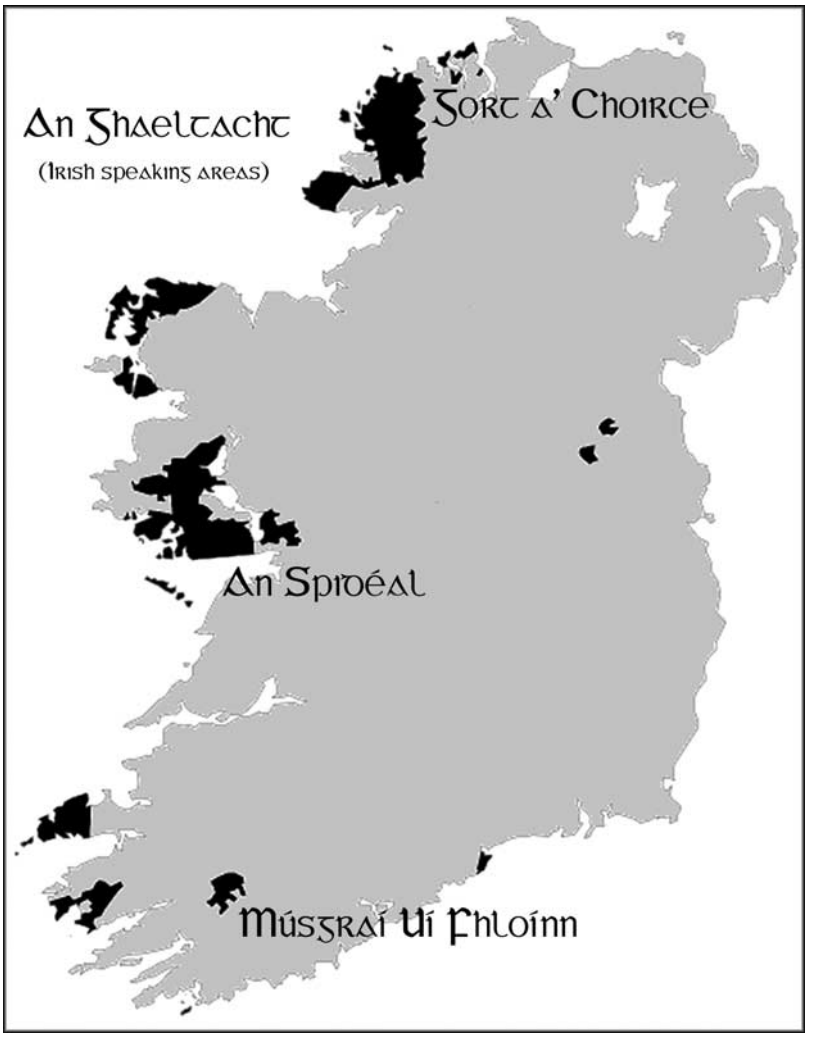

Fig. 3. The geographical distribution of dialects currently supported by Sintéiseoir

9 Editing and storing the concatenation units

Editing the single phones was a straightforward case of isolating the waveform of the phones themselves from preceding and following silence, coughs, lip-smacks *et cetera*.

No formal procedure was devised for editing the half-phones and diphones, other than to excise leading and trailing silence as close to the start and end of the waveform as possible, and to make cuts at phone-centres as consistently as possible, at corresponding points in the waveforms. This became more intuitive and easier as the editing process progressed.

The edited concatenation units were stored as WAV files in a directory called *meain* (media), in the program root on the local file system. Files for each dialect were segregated into subdirectories named after the district in question: *gortachoirce*, *musgrai* and *spideal*. This directory structure is a key aspect of the concatenation process.

https://doi.org/10.1017/S0958344010000054 Published online by Cambridge University Press

10 Developing the software

The software was written in the Java programing language, which was chosen because it is widely supported and platform-independent. A free piece of software called the Java Runtime Environment or the Java Virtual Machine must be installed on a computer (or other device) for it to be able to run Java but versions of this are available for all common computer and mobile operating systems.

Java is an object-orientated language, meaning that that every conceptual entity to which something can be done is considered an object. These objects have certain definable attributes. Thus, for instance, to create a white text field 100 pixels wide by 50 pixels high on screen the programmer must instantiate a text field object and give it those dimensions as attributes. A colour object must then be instantiated, given the attribute of being white, and then designated as an attribute of the text field object.

Well-written Java programs are made-up of of discrete components called classes, each of which has some fixed, defined function. Classes may be compared to the component parts of physical machines.

All coding was done by hand, using the BlueJ development tool running on an Apple Mac and OS X. The software design exactly follows the structure of the framework, with the modules realised as Java classes. No calls are made on external programs.

Generally, a revision was saved every time a significant element of functionality was achieved or a major problem was overcome. In total, 21 revisions were completed.

Since the dataflow through the framework was already known, the basic code that forms the Java class structure was written and saved in its entirety before any program functionality was undertaken. Similarly, the graphical user interface (GUI) class was completely coded before work on any of the others was begun as the required functionality had already been defined; only a very few minor adjustments had to be made to this class later on during development.

All the other classes were coded by working progressively *back*ward through the framework, from playback to tokenisation. This 'back-to-front' approach was taken because although the general input and output characteristics of each module were known, the Java language defines data types formally and quite rigorously and it was found easiest to code each class by deciding what its *input* data format would be, creating requisite functionality to manipulate data of that kind, and then processing the results to conform to a given *output* format, which was always a known factor as the subsequent module in the dataflow had already been coded, and its input data format fixed.

10.1 The GUI

The user interface (Figure 4) provides a text field to accept user input (which initially displays a default message inviting the user to enter some text), a set of radio buttons to select the target dialect, buttons to initiate synthesis and to clear all fields and selections and a text area to display textual feedback to the user.

The underlying code includes a number of filters to reduce or eliminate the possibility of invalid or extraneous data being forwarded to the tokenisation engines: certain graphotactically invalid strings are rejected, as are numerics and other non-alphabetic characters (except for the apostrophe, if located in an orthographically

https://doi.org/10.1017/S0958344010000054 Published online by Cambridge University Press

Fig. 4. Sintéiseoir's graphical user interface

valid position in the string), as numerics and non-alphanumerics are not yet supported by the synthesiser. Leading and trailing whitespace is stripped-out, and any uppercase characters are folded-down to lower case.

The interface will not forward data to the tokenisation engines unless a dialect has been selected. Meaningful textual feedback is returned to the user whenever errors occur.

The class outputs two plaintext strings to the tokenisation engines: that of the user input and also a dialect hint, which is set internally by the program when the user selects the dialect for synthesis.

10.2 Tokenisation

This is the most complicated class in the software, and has by far the most lines of code. Its internal dataflow can be seen in Figure 5. It was decided to create a fork in the

https://doi.org/10.1017/S0958344010000054 Published online by Cambridge University Press

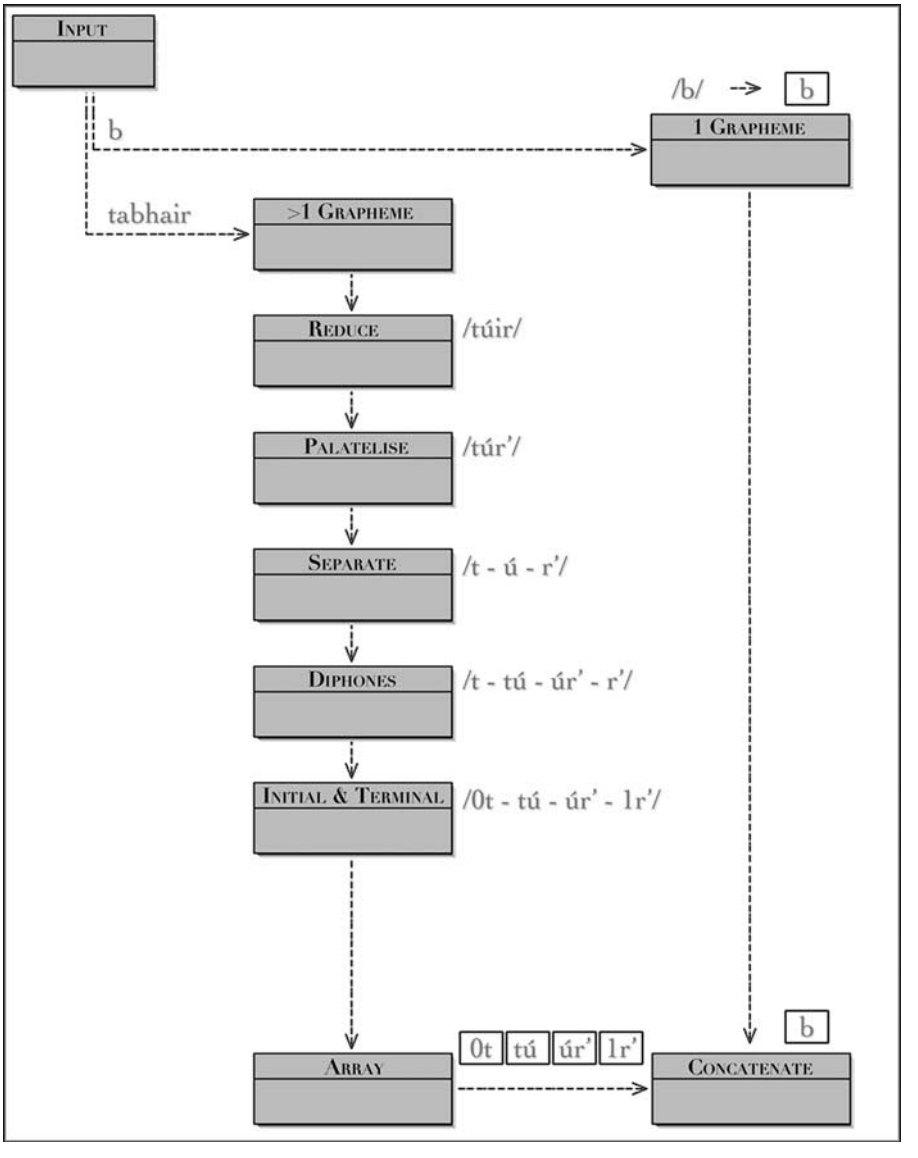

Fig. 5. The main operations performed by the tokenisation engines

dataflow for input strings containing only one grapheme, the reasoning being that most of the tokenisation processing applied to longer strings was redundant in these cases.

10.2.1 Multiple graphemes. User input is parsed to convert the grapheme string to a token string, where the initial and terminal tokens correspond to half-phones and the rest to diphones. This is done in the following series of operations:

(1) The string is reduced by substituting specified substrings with corresponding phone tokens. For this technique to work of course, substrings must be

https://doi.org/10.1017/S0958344010000054 Published online by Cambridge University Press

evaluated in order of length, with the longest processed first. Graphemes that correspond directly to phones are left unaltered. To facilitate the processing of phones whose tokens contain two or more characters (diphthongs and some consonants), these tokens are replaced temporarily by a set of single non-alphabetical place-holder characters.

(2) The string is then inspected to identify consonants and consonant clusters adjacent to the vowel characters "e", "é", "i" or "í", and which are therefore palatalised. Following primes are inserted after each of these consonants. Where "e" or "i" are themselves adjacent to long back vowel characters, and where "é" or "í" are adjacent to short back vowel characters, a glide is indicated, rather than a full vowel. In these cases the short vowel character is dropped from the string as glides are already encapsulated in the concatenation units.

(3) A separation character is inserted between each phone token (palatalised consonants are not separated from their following primes).

(4) The terminal alphabetic character of each phone token is copied and the copy inserted immediately after the following separation character. In the case of the final character in the string, a new separation character is also inserted. Thus, the string is converted from one of phone tokens into one of a half-phone followed by diphones and terminated with another half-phone. Once this process is complete, any place-holder characters inserted into the string at stage (1) are replaced by their corresponding diphthongs and consonants.

(5) The numeric characters "0" and "1" are inserted in front of the initial and terminal half-phones respectively, so that they correspond to initial and terminal half-phone filenames.

(6) The grapheme string has now become a token string, but is still a plaintext object. At this point, however, it is convenient to place the tokens in a binary object called an array, introducing binary data into the dataflow for the first time. An array is a container object of fixed size, wherein stored elements are completely separate from each other. This operation involves splitting the string on the separation characters, which are then discarded.

(7) This array is forwarded to the concatenation engine.

10.2.2 Single graphemes. A small number of Irish words are spelt with a single grapheme. These include "a" (which represents several homonyms) "é" (him), "í" (her) and "ó" (from). Of these, only "a" is not pronounced similarly to the corresponding phone.

As most of the graphemes in the written language correspond directly to an identical phone symbol in the phonetic notation, it was decided that these should simply be placed in a one-element array, forwarded directly to the concatenation engine and uttered as their corresponding phones (velarised, in the case of consonants). Textual feedback is returned to the user interface to inform the user that they are hearing phones rather than words or vocalisations of graphemes. In the case of "a" and the following graphemes, however, it was necessary to map to a different phone symbol:

- The graphemes "ɑ" and "ɑ́" are mapped to the phones /ɑ/ and /ɑ:/.
- The grapheme "á" is mapped to the phone /a:/.

https://doi.org/10.1017/S0958344010000054 Published online by Cambridge University Press

- The grapheme "a" is mapped to the phone /ə/.
  - In this case, the textual message returned to the user interface informs the user that they are hearing a word and not a phone.
- The grapheme "c" is mapped to the phone /k/
- The grapheme "j" is mapped to the phone /d′z′/
- The grapheme "q" is mapped to the phone /kw/
- The grapheme "x" is mapped to the phone /ek′s/
- The grapheme "y" is mapped to the phone /wai/

10.2.3 Output. This class outputs two separate data entities:

- A string array in which each element corresponds to the filename of a soundfile, without the file extension.
- The plaintext dialect hint, which was first instantiated by the GUI. This takes the form "<dialect>/": "gortachoirce/", "musgrai/" *et cetera.*

10.3 Concatenation

This is the module in which plaintext and binary encapsulations of plaintext are replaced completely by binary audio data in the dataflow.

Before describing the concatenation process, it may be helpful to discuss the WAV file format briefly, as this is the medium in which the concatenation units are stored. Most computer literate people with more than superficial familiarity with multimedia will know that WAV files are common, standard containers for storing uncompressed sound; providing high fidelity but at the cost of relatively large filesizes. The reality is more complex, however, as the WAV format permits data to be encoded in various ways – and, indeed, compressed. Therefore, a WAV file created by one device or program may have a somewhat different internal structure from a WAV file created by another. This structure is determined by a series of parameters including sample rate (the number of times per second that the original analogue sound was measured, or "sampled", during digital recording), sample size (the range of memory values made available to store each sample), the number of channels (mono or stereo), signed (the capacity to represent negative integers) and big-endian (the order in which the bytes of data are stored). The values of these parameters in the concatenation unit files must be known in order to manipulate these files. Thus, of course, it was necessary to decide upon and adhere to a specific format to use when recording the human speakers. Also, WAV files are not solely composed of audio data, but of three "chunks" of information: The RIFF chunk which identifies the file as a WAV file, The FORMAT chunk which identifies parameters such as sample rate and the DATA chunk which contains the actual data (samples)" (Niagra College Canada, Technology Division, 2009). The audio data are stored in a series of 'frames'.

When the concatenation class is instantiated it creates:

- An object called a vector into which the concatenation units are to be placed (vectors are container objects, similar to arrays but of variable size).
- An integer object carrying a value of "0".

https://doi.org/10.1017/S0958344010000054 Published online by Cambridge University Press

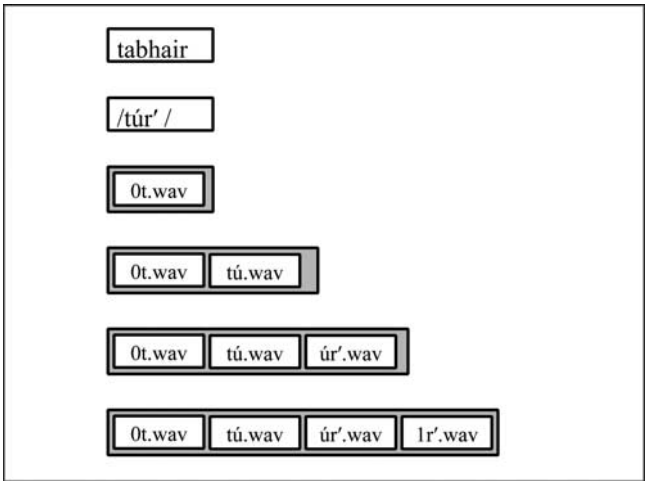

Fig. 6. Concatenation units assembled incrementaly in a vector object

The class iterates through the string array that was forwarded by the tokenisation engine, extracting each element (filename) in turn. On each iteration it performs two operations:

(1) The following temporary text string is concatenated:

"meain/"+<dialect hint> + <filename> + ".wav"

This provides a file path to the relevant concatenation unit: "meain/" references the sound file repository, the dialect hint extends the path to the subdirectory of the selected dialect, the filename references the file itself and the string ".wav" provides the file extension. By referencing the file extension here rather than earlier in the dataflow it only appears once in the code, reducing the scope for error. Also, should the file format be changed for any reason, only this one reference need be updated.

(2) This concatenated string is used to retrieve the file in question. Its audio data frames are then extracted and placed in a new audio data object, called an AudioInputStream. This object is then placed in the vector and the integer object is incremented by the number of audio data frames retrieved.

Thus, with each iteration (Figure 6), the components of the phonic continuum to be synthesised are assembled incrementally in the vector and a record is kept, in the integer object, of the total number of data frames – the frame length.

After the final iteration the vector is passed to an enumeration object, which passes each element in the vector, sequentially, to a SequenceInputStream object. This is the Java object that actually concatenates the discrete elements of audio data into one logical unit. The result is passed, along with the format parameters of the source WAV files and the total frame length, to an AudioInputStream object. This object is then forwarded to the signal processing module.

10.4 Signal processing

As stated, this class has no functionality in this version of the software. It contains only enough code to accept an AudioInputStream and to output it again, unprocessed. In future versions, prosodic processing and discontinuity smoothing at concatenation unit

https://doi.org/10.1017/S0958344010000054 Published online by Cambridge University Press

interfaces *may* be implemented in this class, although it is more likely that two separate classes will be created to implement these operations as they are really different tasks.

10.5 The playback device

This was the simplest functional module to code. It accepts the AudioInputStream object and its format parameters, and performs only three main operations:

(1) The AudioInputStream's data format is retrieved and used to create a DataLine.Info object.
(2) The DataLine.Info object is used to create a Clip object, which is a data line whose audio data can be loaded prior to playback.
(3) The Clip object is then started and the audio data sent to the the host computer's audio subsystem. Thus, the audio output is never saved as a file (although the code could easily be modified to do so) but played aloud.

10.6 UNICODE

As the notational system of the tokens includes several non-ASCII characters (ASCII is a formally defined but basic set of letters, numbers and punctuation marks for American English), operating system and application support for UNICODE was clearly essential. PC, Mac, UNIX and UNIX-like operating systems have, generally, supported UNICODE for years, as has Java, but many of the software (and Web) applications that run on them still do not. A few minor problems related to character encoding were encountered during the project:

- The sound-editing software was unable to save files with names that included non-ASCII characters. Such attempts failed silently, giving the user no indication that the save operation had failed. As a workaround, such files were saved with temporary, one character filenames and then renamed as appropriate within the operating system itself.
- It was found impossible to include non-ASCII characters in filenames when renaming files located on the desktop. This was noteworthy as it did not happen when renaming files elsewhere in the file system, within file browser windows. Since all project files were stored elsewhere in any case, the problem did not affect the work unduly.

11 Results and conclusions

Results were assessed, and conclusions drawn, based mainly on the project's terms of reference. However, some of the broader requirements of the text-to-speech domain were also considered.

11.1 Software functionality

Sintéiseoir 1.0 was tested on the following platforms (fully patched and updated), which were current at the time, and was found to work correctly and consistently:

- Apple Mac OS X 10.4.11 (desktop); Java 1.5.0_13-121
- Microsoft Windows 2000 Professional SP4; Java 1.6.0_05-b13

https://doi.org/10.1017/S0958344010000054 Published online by Cambridge University Press

- Microsoft Windows XP Professional SP3; Java 1.6.0_05-b13
- Ubuntu Linux 8.04 (desktop); Java 1.6.0_06

11.2 Functional vocabulary

At the completion of the project, *Sintéiseoir* 1.0 was able to successfully utter each of the ninety-nine specified lemmata in each of the three specified dialects.

It was decided to further test its current functional vocabulary by submitting to it, individually, the words contained in the thesis abstract. This particular block of text was selected because although the thesis was written in a formal register of the Irish language, it used a large number of common words and words of differing lengths. Also, importantly, the abstract was written at the beginning of the research and was never intended to be used in this manner. Thus, it was considered to stand as valid test material.

The abstract contained 298 words in total. Multiple instances were excluded and because *Sintéiseoir* does not currently support non-alphabetic characters (other than the apostrophe in certain positions), numeric characters and prefixes followed by a hyphen were also excluded. The final inventory was 187 words. *Sintéiseoir* uttered 31 of these correctly, representing a success rate of 16.57%. Of those 31 words, 5 were not on the list of lemmata – 2.67% of the total.

Whilst the tokenisation algorithms will undoubtedly benefit from development, error data returned to the user interface indicated that failures were most often due to the small number of available concatenation units.

11.3 Pitch and duration

As the project's terms of reference excluded work on prosody, no particular expectations existed with regard to pitch or duration and no criteria for success or failure were specified. In the event, the lack of processing for pitch caused no undue problems. Indeed, it was felt that the pitching of *Sintéiseoir*'s utterances was quite naturalistic. However, this was undoubtedly due to its current restriction to single-word utterances and, to an extent, the difficulty experienced by the human speakers in completely removing prosody from their voices whilst recording the carrier material.

The absence of duration processing was found to cause a problem in the case of the neutral vowel (/ə/) however. In Irish, the duration of this vowel is dictated by its position in the word (initial, medial or terminal) and by dialect. Inappropriate articulation is very noticeable, particularly to L1 speakers. For example, a classic shibboleth of L2 speakers is the pronunciation of the definite article "an" (/ən/) with a stressed and relatively protracted initial vowel (/o'n/) when it is correctly little more than a glottal stop.

Six of the ninety-nine lemmata specified in the terms of reference, *agam*, *agat*, *aige*, *acu*, *ansin* and *anois*, begin with the neutral vowel. Without duration processing this vowel was quite protracted in all three dialects when uttered by *Sintéiseoir*.

The first four of these are the prepositional pronoun "by", in the first person singular, the second person singular, the third person masculine, singular and the third person plural – "by me", "by you", "by him" and "by them". This protracted

https://doi.org/10.1017/S0958344010000054 Published online by Cambridge University Press

initial vowel is appropriate for northern and mid-western dialects, where polysyllabic words are generally uttered with stress on the first syllable, and although it is inappropriate for southern dialects, in which the *second* syllable is stressed, it remains intelligible. Therefore, given that prosodic processing was not a project requirement, it was decided to let this pronunciation stand.

However, the initial vowel of the last two words (''there'' and ''now'', respectively) are generally of very brief duration in normal speech, to the point of a glottal stop or even omission, and sounded grossly unnatural when uttered by *Sintéiseoir*. With reference, again, to the absence of a prosodic processing requirement, it was decided to define a short exceptions list for these and certain other words as a workaround, and excise this initial phone completely at the tokenisation stage. The resulting utterances were found to be naturalistic.

It is intended that this workaround be abandoned in favour of appropriate prosodic processing in future versions of the software.

11.4 Intelligibility

As with pitch and duration, no criteria were laid down to assess the intelligibility of utterances generated by *Sintéiseoir*. All successfully synthesised words were found to be comprehensible, despite the existence of audible discontinuities between certain concatenation units.

However, as repeated, long-term exposure will often render the most poorly formed speech intelligible it was decided that at least a rudimentary external assessment of intelligibility was warranted. This was done by playing the fourteen lemmata of more than one syllable (*agus*, *abair, anois, agat, duine, ansin, agam, amach, isteach, muise, aige, eile, éigin* and *oíche),* in each of the three dialects, to six L1 speakers who had not heard *Sintéiseoir*'s output previously. None had any difficulty identifying them, although the discontinuities between concatenation units were mentioned.

11.5 Further development

In addition to the obvious need to expand the single word functionality of the synthesiser, future research will be shaped by the following factors:

11.5.1 An lárchanúint. *An lárchanúint* was only made a consideration in order to reduce the complexity of the project to a level commensurate with an M.Sc., thesis. While it served well enough in this regard, its prescriptive, heavily-simplifying nature proved to be fundamentally incompatible with the overall aim of genuine L1 dialect support. Such dialectical variation as there is arises mostly from variations in the vowel sounds uttered by the human speakers – a freedom only partially permitted within *an lárchanúint* – rather than from the tokenisation process. Therefore, it will have no place in further development of this research.

11.5.2 Concatenation units. Even the perfect speech synthesiser would be of only academic interest without a voice. Therefore, development of the concatenation unit set is, perhaps, the most pressing requirement, it being too limited at this point to

support a speech synthesiser with any practical functionality. The presence of audible discontinuities in existing units must also be addressed by re-editing or, if necessary, re-recording.

Recording the human speakers and preparing the sound-files is a slow and tedious process, and the multi-dialectical nature of the framework multiplies the number of man-hours required by the number of supported dialects – currently three. There is, however, no alternative to undertaking this work as a matter of priority.

11.5.3 The coding platform. Although there is nothing fundamentally wrong with building the program on the Java platform, and delivering it as a locally running executable, the luxury of Web-based delivery, particularly in an age where the Web application is on the ascendant, was keenly felt during testing. Web apps are also far more readily accessible to their target user-base and one need only to look to *abair.ie* to see the inherent benefits of this approach.

In principle, the existing code might easily be adapted to the client/server model, with the user interface converted to a Java applet running on the end user's computer and the rest of the program running remotely on a Web server. However, it is intended that the next version of the software be completely rewritten using XHTML, CSS and Javascript on the client side and Apache Webserver, PHP and, possibly, MySQL on the server side; these technologies, between them, provide the requisite functionality, and form key aspects of the Internet's infrastructure.

References

Buntús Gaeilge (1966), Baile Átha Cliath: An Roinn Oideachais.

Coláiste na Tríonóide, Baile Átha Cliath, Centre for Language and Communication Studies (2008) *abair.ie* http://www.abair.ie/.

Lenzo, K. A. and Black, A. W. (2000) *Diphone Collection and Synthesis* http://www.cs.cmu.edu/ ~ awb/papers/ICSLP2000_diphone/.

Mac Lochlainn, M. (2008) *Forbairt Chreat Ílchanúnach Téacs-go-hUrlabhrtha don Ghaelainn*, unpublished thesis (M.Sc.), University of Limerick.

Niagra College, Canada, Technology Division (2009) *CTEC1631 Course Notes: WAV File Format* http://technology.niagarac.on.ca/courses/ctec1631/WavFileFormat.html.

Ó Baoill, D. P. (ed.) (1986a) *Lárchanúint don Ghaeilge (Téip Léirithe)*. Baile Átha Cliath: Institiúid Teangeolaíochta Éireann.

Ó Baoill, D. P. (1986b) *Lárchanúint don Ghaeilge (Tuarascáil Taighde)*. Baile Átha Cliath: Institiúid Teangeolaíochta Éireann.

Ó Cuív, B. (1944) *The Irish of West Muskerry, Co. Cork*. Baile Átha Cliath: Institiúid Ardléinn Bhaile Átha Cliath.

Ó Domhnaill, N. (1977) *Foclóir Gaeilge-Béarla*. Baile Átha Cliath: Rialtas na hÉireann.

Ó Duilearga, S. (ed.) (1981) *Leabhar Stiofán Uí Ealaoire*. Baile Átha Cliath: Comhairle Bhaloideas Éireann.

Ó Sé, D. (2000) *Gaeilge Chorca Dhuibhne*. Baile Átha Cliath: Institiúid Teangeolaíochta Éireann.

Schmitt, S. R. (2009) Permutation generator http://home.att.net/ ~ srschmitt/script_permutations.html.