REDUCED IMPEDANCE MISMATCH IN SPEECH DATABASE ACCESS

Toomas Altosaar\textsuperscript{1} and Martti Vainio\textsuperscript{2}

\textsuperscript{1}Acoustics Laboratory, Helsinki University of Technology, Finland
\textsuperscript{2}Department of Phonetics, University of Helsinki, Finland

ABSTRACT

Database access can be seen as a series of communicating systems each contributing to the overall efficiency of the query. This paper investigates the effect that impedance mismatch has on speech database access: the efficiency of database transactions between the corpus and the querying application(s). A system where impedance mismatch has been reduced through an object-oriented paradigm is presented and several query examples are shown. The paper suggests that for applications requiring high rates of database access for data with many expressed relationships a tightly coupled object-oriented system extending from the corpus to the end-user application is required.

1. INTRODUCTION

Effective representation of speech databases requires structured models on which spoken language can be modelled. On the other hand, efficient utilisation requires that the speech model be well tuned and tightly coupled to the application which is using the database. The efficiency of the transfer of data between the database and querying application can be measured and is related to the impedance match between two systems.

In general, an effective representation of speech is a prerequisite for efficient database access. However, a clear and structured model where speech information and data can be placed and relations drawn out explicitly does not guarantee that data access will be efficient. Improperized representations of the results of database queries hinder the rate at which applications can access the corpora. For example, queries returning only a textual description of a speech context in an indirect manner such as a file name representing a signal, a pair of indices representing the location of a phone, etc., exhibit a large impedance mismatch if further processing is to take place with other parts of the model structure. Every transaction between the database and application requires a wasteful transformation of representational form.

In previous studies we have looked at utilising the full potential of speech databases with respect to object-oriented modelling methods \cite{1, 2}. Also, relational and object-oriented database management system models for representing speech have been compared \cite{3}. In this paper we present the formalism employed in the QuickSig Speech Database System \cite{4} for addressing impedance mismatch between speech corpora and querying speech applications.

Instead of database queries returning depleted descriptions of a speech context or event, live pointers to the actual objects of the speech structure are returned. This allows for applications to gain immediate and direct access to the speech model structure.

Several different speech corpora, e.g., TIMIT, Kiel, ANDOSL, BAS, etc. have been modelled in QuickSig in a generic manner. Applications that utilise these corpora include speech synthesis, recognition, and speaker verification-recognition tasks. In this paper, examples of database transaction-intensive processing are given, e.g., via recursive database queries where the results of queries determine further processing.

2. IMPEDANCE MISMATCH

In many different types of systems, e.g., electrical, systems exhibit a characteristic impedance. Electrical impedance is defined as a measure of the total opposition to current flow in an alternating current circuit. Made up of two components, ohmic resistance and reactance, impedance is represented as a complex value.

In general, an analogy can be drawn from the physical realm to the software realm for defining the impedance of software modules. The terminal input and output characteristics of a module need to be measured. Whenever two software systems communicate with each other a transformation of representation may be required. This is wasteful since time must be expended to map from one representation, protocol, etc. to another. Impedance mismatch occurs in databases when a query language such as SQL is embedded in a programming language and the objects that are manipulated by the query language statements are not subject to the type checking constraints of the language \cite{5}. To ensure the most efficient transfer of information the impedances of the communicating systems should be identical.

Figure 1 shows a series of software modules used to communicate with a speech corpus. At the left is the corpus in its media-distributed form. The next level is a database access interface to the speech corpus which provides a mechanism to retrieve data. The database access level may provide a level of abstraction that facilitates usage, e.g., instead of referring to file names in the corpus, an object-oriented representation may exist so that pointers to objects can be used instead. The remaining modules are a sequence of one or more querying
applications that communicate with the database access module. Between each querying application there may exist a necessity to change the representation form of the data, e.g., at a low level (near the speech corpus) a set of indices may be used to represent the sample values in a signal at some point in time. However, at the querying application level a complex, high-level object may be represented instead, e.g., a human uttering a sentence. Mapping between different representations during database access reduces the efficiency of the system — to reduce impedance mismatch a system should strive to use the same representation throughout all processing levels. Therefore, if high-level, complex data structures are used in the final querying application then the use of an object-oriented model throughout all layers is advocated.

3. DATABASE ACCESS IN QUICKSIG

In an object database management system (ODBMS) high level structures and operations can be used directly in the query. This gives the end user, e.g., the final querying application (in the chain seen in figure 1) a fine level of control over query formulation.

Database access in QuickSig operates in a tightly coupled environment. Signals, functions, processes, talkers, annotations, etc. are modelled as objects. Class inheritance is used extensively which promotes code reuse as well as robustness. Polymorphism is supported by generic method functions which allow a single program entity to refer at runtime to instances of a variety of types or classes [5]. Lisp along with its Common Lisp Object System (CLOS) is the language of implementation used in QuickSig and provides for an efficient and extensible language onto which software modules can be designed and implemented.

Not only are the communicating software layers and modules tightly coupled but the representation of the speech is already compiled and linked. The latter relaxes the requirement for repeated parsing and interpretation of data objects during the search phase. Thus searches can be executed quickly and efficiently.

A diverse variety of speech database formats are handled within QuickSig, as can be seen in figure 2. This is achieved by modelling different corpora in a generic manner. Corpus specification modules (CSM) are defined for every different speech corpus which is to be used. A CSM adds missing information to the speech corpus, e.g., data types, structure, etc., that would otherwise be required to be specified by specific program code or human intervention at query time. CSM supplied meta-information along with a corpus allows the corpus parser to access any part of a corpus intelligently.

A compiler and linker use information supplied from a set of resources that include knowledge regarding character sets, phonetic alphabets, annotation styles, etc. to transform data from corpora into linked object structures called frameworks. Figure 3 shows the visualisation of a framework containing the acoustic, phonetic, linguistic, and orthographic planes, as well as sentence, word, syllable, and phone/phoneme levels.

Once a framework has been constructed for each utterance in a database, searches can be carried out. Performing database queries on a highly integrated object structure where objects possess specialised knowledge of their type and where links to other objects are explicit, allows for specific and robust query expressions. Efficiency as well is much improved since string searches, file name mapping, etc. are avoided altogether, which might occur during database query time in a standard, relational database management system (RDBMS).

4. EXAMPLE QUERIES

In this section examples of queries are presented. The corpora consists of 1000 sentences spoken by a single male Finnish speaker. Annotations in the phonetic domain include the following levels: segment, phone, syllable, word and sentence. Morphological analyses of the material is also present and allows linguistic and phonetic domain words to be queried as to their part-of-speech, tense, voice, etc. Prosodic information in the form of associated pitch signals is available and can be queried/calculated via the function \texttt{unit-F0} that can be applied to any sized unit, e.g., the average F0 value for a syllable can be found by the following form:

\begin{verbatim}
  (unit-F0 unit)
\end{verbatim}

where \texttt{unit} is a syllable unit existing in a compiled and linked framework. \texttt{unit-F0} returns a single real value for the average F0 over the unit. Additional arguments to limit the relative size of the interval can also be supplied to \texttt{unit-F0}, e.g., if only the middle 1/3 of the unit's interval is to be used.

The search engine in the QuickSig ODBMS is invoked by calling the function

\begin{verbatim}
  (find-units unit :test test)
\end{verbatim}

The \texttt{find-units} generic function accepts almost any type of speech object: a single framework, a list of frameworks, a signal, a talker, etc. Since all objects within the database are heavily linked a route is always available from any object to another. Thus the annotation(s) can be found with respect to an utterance, and the framework(s) located. If a talker is supplied as the argument then the default is to apply the test to all utterances the specified speaker has uttered in the corpus.
Figure 2: System used in QuickSig to minimise impedance mismatch between corpus and querying application. Each different corpus has a corpus specification model (CSM) defined for itself. The linker and compiler utilise resources to generate an object structure for each utterance in the database. Database access occurs in the same highly integrated environment.

Figure 3: Multi-dimensional representation framework for a sentence in QuickSig. Objects within the planes (domains - acoustic, orthographic, linguistic, phonetic) are speech units, e.g., phones, syllables, words, sentences, and are linked to each other as well as other units in different domains.

The function find-units accepts as a keyword argument test which can be a predicate function. When test is applied to individual units within a framework it returns either T or NIL, Lisp's indicators for true or false. test need not be a predicate function but can also be a function that returns a fuzzy value, e.g., a real value in the set [0,1], that indicates the closeness of a match.

Finally, find-units returns as a list the actual objects existing in the framework that matched the predicate function. These objects can then be sorted according to some criterion, if desired and presented to the user in a number of ways.

4.1. Example of Morphological Query

In this example we wish to find all of the words in the database that are a copula and where the next word is an adjective. Formulating this query in Lisp using the find-units function is as follows (in this case *abstract-node* is a user defined node selected from a database grapher specifying which set of data to apply the query to):

```lisp
(find-units *abstract-node* :test #'(lambda (x)
  (and (typep x 'phonetic-word-unit)
       (copula-p x)
       (adjective-p (next-unit x))))))
```

In this example x represents any unit find-units applies test to. For the test to return a non-nil value three conditions must be met: 1) the current object must be a phonetic word unit, 2) x is a copula, and 3) the next unit with respect to x is an adjective. As can be seen, this high-level query deals with the same objects found in the representation frameworks and no mismatch in impedance occurs. Objects returned by find-units, in this specific case 63 words, can be used for further
4.2. Example of Recursive Query

In this example we desire to find the average unit rate of a phone in its actual context, i.e., to determine the number of phones in a window centred around each phone. As a default, the window size is 1 second. Therefore, starting at some phone we need to traverse backwards in time finding how many phones (or portions thereof) are within the 500 ms window. A similar process is required to process the phones forward in time. This type of query is data dependent since the number of phones will vary and is well solved using a recursive query, as shown below. As can be seen, the generic function unit-rate defines two mutually recursive functions, find-left-edge and find-right-edge (shown in bold) using the labels construct. These functions perform the actual traversal from unit to unit until a window edge has been reached, updating the local variables as needed. Finally, when the edges and number of units have been found, the average unit rate is calculated.

When applied to the 692 plain sentences of the 1000 sentence corpus, 44093 phone units had their average phone rate calculated over a one second window. Represented as a histogram, the distribution of average phone rate can be seen in figure 4.

![Distribution of average phone rate (units vs. phones/second) calculated over 692 sentences.](image)

Even though unit-rate utilised two recursive functions, the functions prev-unit and next-unit are defined recursively as well. The number of calls to these functions were measured during the repeated queries and were 458427 for prev-unit and 457978 for next-unit. Total execution time on a 300 MHz PowerPC G3 based laptop was 54 seconds which translates to over 8000 framework queries per second. Without a precompiled and linked framework structure performance would be orders of magnitude slower. Even worse performance would be obtained if the querying applications existed as different applications running on the same machine and I/O would be through the file system.

5. SUMMARY

Problems associated with querying speech corpora in terms of database impedance mismatch were addressed. To minimise impedance mismatch identical data structures are required at all levels of the query process. Thus an object-oriented model was advocated. QuickSig — a speech processing and annotation system based on object-oriented programming — was described. By utilising speech objects within the queries a good impedance match was found between the high-level query and the actual corpus. By operating with the same objects throughout high and low levels of query, impedance mismatch is kept at a minimum — and thus database access performance is maximised.

6. REFERENCES