# Design of an Interactive Web-based User Interface for Speech Database Query Formation

**Toomas Altosaar & Matti Karjalainen**

Laboratory of Acoustics and Audio Signal Processing
Helsinki University of Technology, Finland
Toomas.Altosaar@hut.fi Matti.Karjalainen@hut.fi

**Abstract**

This paper presents the design of an advanced experimental system currently under development that allows speech database queries to be defined in a high-level manner both intuitively and interactively. Queries are graphically specified by creating matching template units that are instances of classes, e.g., phones, phonemes, syllables, and words. Also, the features and properties of these instances are defined, e.g., voicing, part-of-speech tags, etc. Types, features, and properties of each unit can be made as specific or general as required, enabling different degrees of query selectivity. Relations between template units are used to specify query contexts and are indicated through links. Template units can be named symbolically, enabling the formation of reusable libraries of queries. Interaction is promoted since formed queries can be applied to object-oriented speech databases with immediate feedback. Databases can be located on a central server or on a local machine. Being html-compliant, the system is accessible from different web-browsers and platforms making it a portable tool for speech processing.

## 1. Introduction

Speech corpora form a solid basis for research on spoken language since theoretical hypotheses can be tested comprehensively on large volumes of data. However, before corpora can be used an infrastructure is necessary that allows access to the data. A speech database – corpora integrated with an access mechanism – fulfills this requirement (Hendriks, 1990).

Speech database access is typically performed on relational database management systems (RDBMS). Here the rich structure of speech first needs to be flattened due to the inherent representational constraints of the RDBMS paradigm. In an object-oriented database (OODB), a more natural model of speech can be formed thus allowing complex queries to be designed and applied reliably (Altosaar, Millar & Vainio, 1999).

Typical queries in an OODB system may take the form of functions that search the local environment of a speech utterance, moving repeatedly from unit to unit and testing for matching contextual locations. These query functions may return matches according to some binary or continuous valued function, e.g., all [A] phones in some context are returned as a set of matches, or, all units that the query was applied to are returned but sorted according to some closeness-of-match criterion.

However, specifying these query functions textually, such as with computational methods in a programming language, e.g., C, Lisp, etc., usually requires that the user be fluent in the query syntax or implementation language of the speech database. This has the negative effect of restricting the number of potential users being able to utilize the system. An intuitive and easy to use interface would have much value since design and manipulation of speech database queries could be performed by non-programmers. By lowering the required skill threshold, meaningful research could be performed, e.g., by students and people working outside the core speech technology area. Furthermore, since the system described here is designed to work on html-compliant web-browsers, it would be highly portable, functioning on different computer and operating system platforms. Other benefits would include the possibility to test and verify the related annotations of a corpus since queries could be formed interactively and tested immediately on parts or entire volumes of speech data. Syntax and validity checking of queries by the system is also important since faulty queries could be detected before their expensive application to large databases would commence.

This paper presents the design of an interactive web-based system for forming and manipulating queries that can be applied to speech databases. Through visualization, the reliability of query specification is expected to be enhanced in the sense that fewer human errors are made when compared to writing equivalent query functions by hand in a textual format. The interface under development is integrated with the QuickSig speech OODB system (Altosaar, 2001). Speech corpora that have been represented under QuickSig include, e.g., TIMIT, ANDOSL, Kiel, Estonian and Finnish.

## 2. Object Model for Representing Speech

### 2.1. Representation Frameworks

Representing data with objects allows abstraction layers to be created that hide implementation details. Focus can move from low-level issues to higher-order concepts that are more directly related to the problem at hand.

Speech offers itself to be modeled readily with objects based on classes, e.g., classes can be defined for items such as a signal, phoneme, segment boundary, a talker, etc. Objects, which are instances of these classes and strive to model actual data, can be associated with other objects, e.g., a talker may have produced a set of signals, with each signal having one or more annotations, which in turn may include sentence, word, syllable, and phone objects. Objects can be made aware of their local surroundings using links enabling efficient automated inferences to be performed during query stages.

A collection of speech objects modeling a related phenomenon is referred to as a *representation framework* (Altosaar, 2001). A *domain* is one part of a framework and can readily be used to express a language theory where different hierarchical *levels* exist, e.g., in phonetics the creation of sentence, word, syllable, and phone levels for

some available data may be a natural solution. Each level in turn may have a set of specialized speech objects called *units* associated with it that represent actual occurrences in an utterance, e.g., some actual phones.
Figure 1 shows a 3-D graphic visualization of a representation framework for an utterance of speech that includes acoustic, phonetic, linguistic, and orthographic domains. Each domain, depicted as a plane, has specific levels and units associated with it. Such a model of speech can be queried readily since the types and relations of objects to one another have been pre-computed.

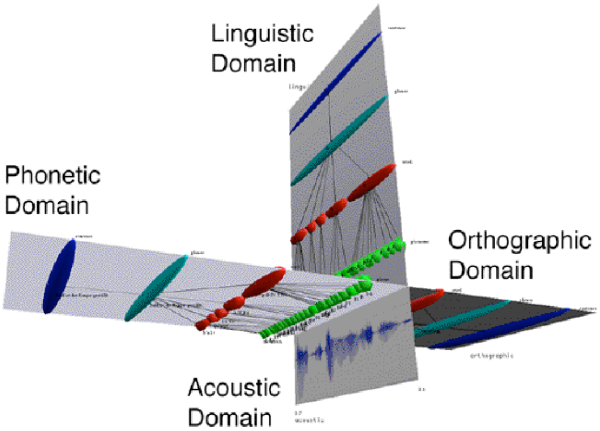


*Figure 1: A representation framework residing in an OODB that models an utterance of speech. Queries operate on speech units and their relations with units in similar or other domains.*

## 2.2. Queries

Search spaces are typically sets of frameworks, e.g., an entire corpus, or several corpora. Matching contexts can be found in a representational framework by first defining a query template that resembles a fragment of the structure shown in figure 1. Database search takes place by measuring the closeness-of-match between a query template and local framework structure. For each comparison a distance measure is calculated that indicates how well the template matched part of the framework. Typically, the distance measure returns a simple binary valued result: either the query template fits into to the structure at some location or it doesn't. The result of a query is a set of matching locations represented as a list of pointers to speech units in the frameworks. These can be used in further speech processing tasks efficiently since they offer a low database impedance mismatch (Altosaar & Vainio, 2000) with the application, i.e., the results of the query and the speech processing application share the same data structures.
This process is best illustrated through an example where the goal is to locate all fricative-vowel phone pairs in one or more frameworks. This query can be envisioned as a simple dipole query template structure, consisting of the two phone units shown in figure 2.


*Figure 2: Example query template for identifying locations of fricative-vowel [F-V] pairs.*

For an example search space, figure 3 shows a partial framework depicting the phonetic annotation for the Finnish sentence "isompi on suurempi", where sentence, word, syllable, and phone speech units exist.

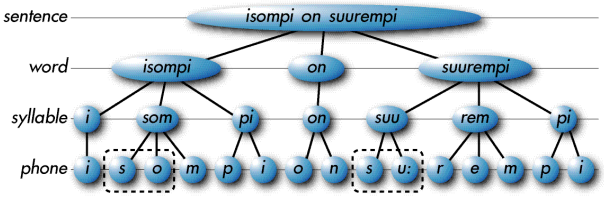


*Figure 3: Part of a framework for an example utterance in the phonetic domain. Boxed areas show matches for the query template of figure 2. (Finno-Ugric phonetic alphabet is used on the phone level; others are marked with orthography).*

By visually inspecting the query template and the framework to which it is applied, one notices that two locations fit the match: the [s-o] and [s-u:] pairs. This is since [s] can be thought of inheriting from the more general fricative class, and both [o] and [u] from the vowel class.

## 2.3. Query Expressed in Lisp

In order to automatically locate desired locations in an utterance of speech, small sections of programming code have been written to model these queries. This code is then applied to a framework's existing units, or a subset of these, and an index calculated indicating how well the query matched the framework at some location.
For example, in Lisp, the language used in the QuickSig OODB system, the query of figure 2 can be written as:

```
(lambda (x)
  (and (typep x 'fricative)
       (typep (next-unit x) 'vowel)))
```

Here *lambda* defines a function and the variable *x* represents an arbitrary unit in the framework where query focus is currently positioned, i.e., the unit being currently tested. The *and* macro requires that all of its arguments are not *nil* valued for it to return a non-*nil* value (*nil* signifies false in Lisp). The *typep* function tests whether its first argument inherits the class specified by the second argument (a symbol) in its class hierarchy. For example, the desired context requires the first part of the phone pair query to be a fricative, so the variable *x* is tested whether it has the class *fricative* in its class hierarchy. (It should be noted that the QuickSig speech OODB system relies heavily on class hierarchies and automatically builds frameworks prior to query application (Altosaar, 2001)). For the template to match part of the framework, the second phone must be a *vowel* and this is achieved by testing the *next-unit* of *x*. Predefined functions exist for units in the frameworks that can be applied generically and utilize the links in the frameworks to access local context. For example, the functions *prev-unit* and *next-unit* access adjacent units on the same level, the function *units* returns a list of all lower-level units, the function *unit-of* returns the parent unit(s), etc. These functions also accept additional arguments for inter-domain navigation.

However, as the complexity of the query increases so does the complexity of the textual query. This can be seen if we add the following two conditions to the query template of figure 2:

- the phone pair should exist at the beginning of a syllable (refer to this syllable with variable *syllx*)
- *syllx* is neither a word-initial nor a word-final syllable.

Figure 4 shows the modified query template required to express these two additional constraints.

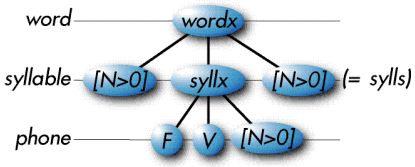


*Figure 4: Original query template with additional constraints. Units marked with bracket notation are wild-card units, e.g., at least one syllable unit must be present before and after syllx, and at least one phone must follow the [F-V] pair.*

By looking carefully at figure 3, it can seen that the new query continues to accept the first match from the original query, but rejects the second match since both additional contextual constraints are not fulfilled by the [s-u:] pair.
The program code that represents this new query template can be expressed as follows:

```
(lambda (x)
  (and
   (typep x 'fricative)
   (typep (next-unit x) 'vowel)
   (let* ((syllx (unit-of x))
          (wordx (unit-of syllx))
          (sylls (units wordx)))
     (and
      (eq x (first (units syllx)))
      (neq syllx (first sylls))
      (neq syllx (first (last sylls)))))))
```

The initial part of this query is similar to the one for figure 2. Since access is needed to all of the syllables in the word, the *let** construct sets up three temporary variables: *syllx* is a syllable in which the potential [F-V] pair resides, and syllx is then queried for its parent *wordx*. The variable *sylls* is a list of all syllable units within *wordx*. With this information, an additional *and* form is now able to check for the additional constraints: the function *eq* is a predicate function that checks whether *x* (now known to be a fricative) is the same object as the first unit in *syllx*, and that *syllx* should not be the first syllable in the word, nor the last one. The *let** construct finally returns the result of the additional constraints (either a non-*nil* or *nil* value) and thus participates equally with the initial [F-V] constraint. When applied to the framework in question, this time only the [s-o] phone pair is found to match the new query and a list containing one element is returned: a pointer to the first [s] phone unit in the framework. The reason why [s] is returned and not a word or a syllable unit is because query focus was specified as being the fricative.
As seen by this example, defining queries in the implementation language of the OODB allows for high degrees of expressive freedom. Complex queries, e.g., those including recursion, or structural elements that are defined to be of a wild card matching nature, can also be written in textually represented program code. Some criteria that we have found to be important and existing in a query formalism is the ability to:

- access local context through the use of navigational functions, e.g., *next-unit*, etc., that utilize explicit or logical links in frameworks,
- associate symbolic names with query templates so that libraries of queries can be published and reused by other queries and users,
- define local functions within a query so that recursive search methods can be applied.

This textual formalism is however error-prone to the programmer and cryptic to the non-programmer. Learning to use a specific textual query syntax or programming language presents a substantial usage barrier in any case. If query structures could be created visually, e.g., on a standard html-compliant browser, then this would improve robustness, address the learning-curve issue, and provide a portable implementation base as well.

## 3. Design of the User Interface

Retaining the expressive power of textual queries in a user interface requires several key components. These elements must enable and aid the user in forming the necessary constraints in several different query dimensions, e.g., structural, type, and property. The function and design of the components of a query formation interface are now presented.

### 3.1. Structural Editor

This part of query specification is used to generate only the query's structural shape, i.e., refer to only the shapes of figures 2 and 4. The user begins by first selecting a domain and level from a menu and then creates a single unit. Parents, siblings, and children can then be subsequently added to this unit or other units to create the desired template structure. Query template units can be defined over many domains and linked to one another if necessary to form multi-dimensional queries (Altosaar, 2001). For this reason it is advantageous that the graphical interface of the structural editor support 3-D visualization. The left half of figure 5 shows one possible user interface for a structural editor. A single mouse click either selects or de-selects a unit, e.g., unit F is currently selected. Unit V is set as query focus.
A symbolic name can be assigned to a selected unit and published so that other query structures can reference it. Named queries can be combined and modified freely to form new queries that can be used by other users. For example, if the user saved the [F-V] phone pair query of figure 2 into a library, this could be attached to unit *syllx* of figure 4 in one editing step.

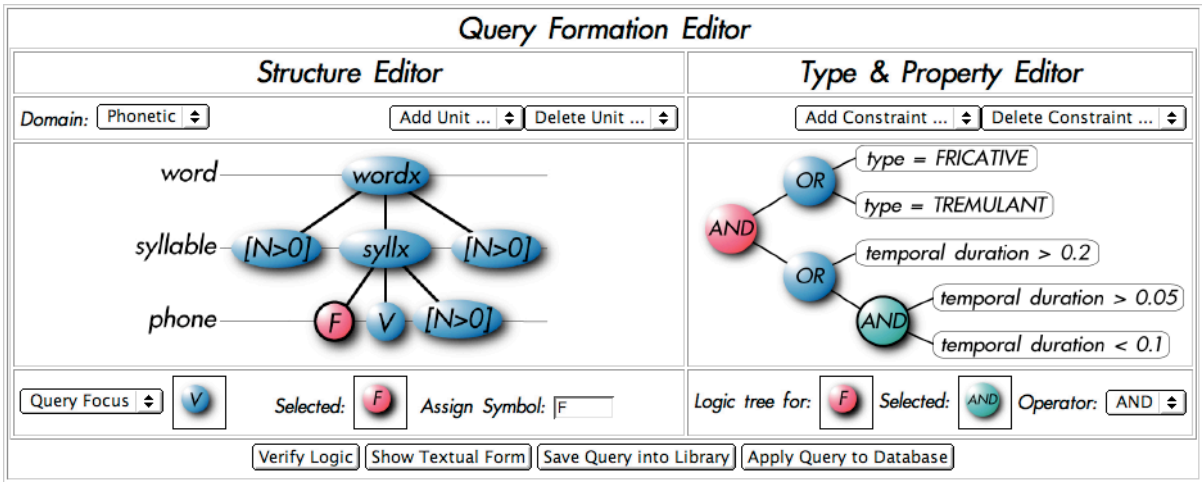


*Figure 5. Depiction of a possible user interface that is designed to fulfill design constraint specifications.*

### 3.2. Type and Property Editor

The internal behavior of template units is defined through a type and property interface. For example, a unit may be required to be a phone such as [A], or a less specific unit such as a back-vowel, vowel, or any phone. Types and properties are assigned to a unit by constructing a logic tree of any breadth or depth. Required query complexity can thus be captured, see the right side of figure 5. Operators can be any Boolean or analog function.

Multiple constraints may be assigned to a single query unit as well, e.g., a template unit may be specified to be a fricative *or* a tremulant. Likewise, temporal constraints can be specified when appropriate. For example, the fricative unit in the previous example can be assigned any number of absolute or relatives time constraints, e.g., its temporal duration can be required to be disjoint: a) between 50 ms and 100 ms, or b) greater than 200 ms, as seen in figure 5, or a certain percentage of another unit's or structure's duration. Quantity of phonemic units can also be specified, e.g., short, half-long, or long. Other properties of units, such as part-of-speech tags, can be assigned to applicable units in a similar manner.

### 3.3. User Interface

Figure 5 shows one of many possible designs for an interface that enables users to generate queries in a high-level and intuitive manner. While queries are being formed they can be applied to a database immediately and the number of matches indicated to the user. Matches can be passed on to other speech processing tasks directly, e.g., speech synthesis or recognition algorithms, without the need for inefficient temporary file-system operations.

## 4. Interaction with Server

For the system to be portable and function in a client independent manner the server performs all computations. For example, mouse-clicks in graphical windows, selections via pop-up menus, etc., are relayed back to the server that keeps track of interface state. This information is converted into Lisp forms, compiled, and is passed on to a search engine which gives immediate feedback to the user. Integration between the OODB search engine (QuickSig) and the http server software (cl-http (Mallery, 2003), portableaserve (Sourceforge, 2002)) is high since both applications run in the same Lisp application environment. Inter-application latency is therefore low and only network delays affect the work rate of users. The latter can be reduced to a minimum by having the browser and OODB/http applications run on the same physical machine.

## 5. Conclusions

This paper described the design of an advanced user interface for forming speech database queries. Query robustness, ease of use, and portability issues are addressed. Currently we are investigating standard methods to implement the interface and are considering portable techniques such as forms, CGI, and client-side image maps applied to the interface elements, and 2-D SVG (W3C, 2003) and 3-D OpenGL (Silicon Graphics, 2004) rendering of query templates.

## 6. Acknowledgements

Funding from the Finnish Academy is gratefully acknowledged for the project entitled "Integrated Resources for Speech Technology and Spoken Language Research in Finland" (project no. 53005).

## 7. References

Altosaar, T., Millar, B. & Vainio, M. (1999). Relational vs. Object-Oriented Models for Representing Speech: A Comparison Using ANDOSL Data. In Proc. of Eurospeech-99 pp. 915-918. Budapest, Hungary.

Altosaar, T. & Vainio, M. (2000). Reduced Impedance Mismatch in Speech Database Access. In Proc. of the ICSLP-2000. vol. 1. pp. 778-781. Beijing, China.

Altosaar, T., Object-based Modelling for Representing and Processing Speech Corpora. Report no. 63 / Helsinki University of Technology, Laboratory of Acoustics and Audio Signal Processing. Espoo, Finland, 2001.

Hendriks, J. (1990). A Formalism for Speech Database Access. Speech Communication, 9, pp. 381-388. Elsevier Science Publishers B.V., North-Holland.

Mallery, J.C. (2003). CL-HTTP Common Lisp http Server. ftp://ftp.ai.mit.edu/pub/users/jcma/cl-http/devo/

Silicon Graphics Inc. http://www.opengl.org

Sourceforge (2002). Portable AllegroServe http://portableaserve.sourceforge.net/

W3C. Scalable Vector Graphics (SVG) 1.0 Specification. http://www.w3.org/TR/SVG/