A Software System for Phonological Data Acquisition and Analysis

Gregory Hedlund and
Philip O'Brien

Department of Computer Science
Memorial University of Newfoundland

A joint dissertation submitted to the Faculty of Science in partial fulfillment of the Bachelor of Science Honours in Computer Science (Software Engineering)

April 2004

Abstract

Comprehensive linguistic software is difficult to find. Very little work has been done to design software that is easily integrated into other systems or provides a channel for linguists to share their research data. This is of primary concern to linguists who wish to obtain data from other researchers or make their own readily available.

Providing extensible linguistic software can result in very powerful tools being incorporated into a single application. Consequently, specifying and designing a succinct software package that allows fast and effective collection of data and processing of this raw data into analyzable elements can become very involved. The architecture designed in this thesis promotes easy addition of future linguistic software modules, resembling plug-ins. This design is based around linguistic experiments and linguistic data corpora organization.

Linguistic experiments are performed on databases of potentially thousands of records. An XML database is an elegant solution to storing structured linguistic data and exporting this data to other systems and applications. Searching such a database for structures that exemplify specific search criteria is also necessary. The query language developed within this document was designed specifically for linguists: incorporating terms from linguistic study to avoid steep learning-curves, allowing user-defined operations on data for more powerful extensibility, and an intuitive syntax for ease of contruction.

These features can prove to be critical in accelerating linguistic research and fostering dependable experiment environments in linguistic study.

Chapter 1: Introduction

The study of language acquisition has been the focus of linguists for several decades. An understanding of how and why children develop linguistic and of the patterns for speech production observed throughout developmental stages is of great interest.

Because research in language acquisition is typically based on tens of hours of monitoring and subsequent transcriptions, an effective method of data storage, information exchange, fast and efficient analysis of large corpora and a robust system to interface these requirements must be established. Attempts have been made to provide such an environment in the past. One of the greatest contributions comes from the *Child Language Data Exchange System* (*CHILDES*) project. This system provides much functionality for determining trends or frequencies within a set of data. This will be discussed further in Section 2.2.1.

However, the tools available through *CHILDES* focus primarily on morphological and syntactic aspects of language acquisition [15]. In contrast to this, this thesis focuses mainly with the development of software for phonological analysis of language acquisition and finding a means to store, retrieve and exchange phonological data. *ChildPhon,* an application that will be discussed in Section 2.2.2, provides much of this functionality and sports a user-friendly interface. However, in its current implementation, *ChildPhon* is neither portable nor efficient [15].

In this thesis we seek to formalize an effective means of storing and exchanging data on phonological language acquisition, to explore a more efficient method of analyzing these data, and to implement these functionalities in a succinct software system. Multimedia support and cross-platform portability are also addressed.

From a brief synopsis of available resources, as they relate to computational analysis on phonological data, no systems have been found that provide the needed services targeted within our software's specifications. A well-specified and engineered system will be presented in this thesis to provide a solution to this problem.

The demand for a software system that supports the needs of linguistic research is real. Linguistic software is scarce meaning the list of requirements that must be investigated and refined is long and its complexity great. Much of the functionality that will be detailed in this thesis has not been considered from a software engineering perspective. The system's functional and non-functional requirements will be formulated from the elicitation of the client's own requirements.

We begin this document by laying a foundation for understanding much of the information that will follow in this document. Phonological process, XML and linguistic query languages will be covered here. This brief query language survey will be used in evaluating the *linguistic* query language specified in this document.

Chapter 3 begins with a discussion on the feasibility for the application. It then defines the requirements necessary for the system and ends with a listing of use cases and a conceptual model. We begin our system design in Chapter 4 with a brief description of two of the more important algorithms used: Syllabification and Alignment. Section 4.3 describes a query language that will be used within the application. Chapter 4 then finishes with a small discussion on system reports. Once our system design is given, we propose our additions to an existing XML schema that will be discussed in Chapter 5. In Chapter 6 we display the software architecture defining each subsystem in the application. We conclude the thesis with a look ahead on the future work applicable to this project.

Chapter 2: Background

In order to provide the necessary background for the key issues presented in this thesis, we first introduce some fundamental information that will help in understanding the problems addressed and the solutions proposed.

Background knowledge pertaining to phonetics and phonology is introduced first, followed by an overview of some of the previous attempts at implementing a software design for analysis and exchange of phonological data. We then turn to a more technical subject matter concerning an *eXtensible Markup Language* (XML) and conclude this chapter with an overview of some of the more prominent and powerful database query languages for linguistics that will apply to our discussion later in this document.

2.1 Phonology

Because phonology is a grounding aspect of this thesis, we will now introduce some background knowledge in this area.

Phonology is the study of the organization and mapping of the cognitive representation (within the mental lexicon) of languages' sound systems. Sounds, syllables and words are stored and organized in a speaker's brain and are translated into sounds and utterances verbalized by this speaker during speech acts.

In the following sections, we explore some of the fundamental concepts of phonology: sounds and their transcribed representations, which incorporate an internationally-accepted character set used to conduct research in the areas of phonetics and phonology, syllables and syllable structure, words and sentences.

As we will see later, research in phonetics requires pair wise comparisons between target form, i.e. the forms that the learner is attempting, and actual forms, i.e. the

learner's rendition. At the end of this section we present an overview of a method used in phonological research to more accurately compare pairs of syllables. This process is called alignment. Alignment is a method enabling systematic comparisons between phonological forms to facilitate research in phonological acquisition. Corresponding applications, such as a suggestive automated alignment, are consequently discussed.

2.1.1 IPA Character Set and Feature Sets

Languages of the world utilize different sounds and sound combinations. To attain a uniform characterization of these sounds, linguists use a common, internationally-accepted, character set. This symbol dictionary, as sanctioned by the International Phonological Association (IPA), is a character set that represents the sounds attested in natural languages.[1]

A one-to-one mapping exists between sounds of the world's languages and the IPA characters. This enables the **transcription** of verbal utterances into equivalent IPA string representations. Research based on phonetic transcriptions can thus indirectly focus on the characteristics of the sounds produced; more on this in Section 2.1.2.

For example, the English words 'blanket' and 'screaming' can be transcribed as [blæŋkət] and [skrimɪŋ] respectively. Within [blæŋkət] we can look at the fourth character, the IPA symbol /ŋ/. This symbol is mapped to a **voiced** consonant exhibiting a **velar** place of articulation and a **nasal** manner of articulation. This information is summarized in Table 2.1. It differs from the sound mapped to /n/ in that the former would be described by non-linguists as having the same place of articulation as the first sound in 'get' (velar). When pronouncing such words as 'branch', one will notice the absence of any 'g' articulation associated with the 'n'. Contrast this to carefully pronouncing 'blanket' or 'string' (the latter can be transcribed as [strɪŋ]). The different pronunciations between the 'n' of 'blanket' and the 'n' of 'branch' are caused by the

[1] Many sounds that can be produced by the human physiology are not formally incorporated into any languages or are merely non-sense sounds (i.e. gaseous reflux). Such sounds are not incorporated in the IPA character set.

sound that follows the ‘n’ in each word. As the sound-producing system usually anticipates the next sound in the speech stream, preceding sounds can be influenced. Phonetic transcriptions are more relevant to phonologists than *orthographic* – the spoken language spelling – transcriptions because they more precisely convey information about the sounds of words and the features of these sounds.

Each IPA character symbol in our set has a list of binary features denoting the fundamental attributes of sound, i.e., positive (+) for voiced vs. negative (-) for unvoiced, positive (+) for sonorant vs. negative (-) for non-sonorant. This list is called the **feature set** of the IPA symbol. Returning to the [blæŋkət] example, we can view the feature set for each character as shown in Table 2.1. Each symbol has a unique set of positive values. While the presence of some features implicitly suggests the presence of others ([approximant] implies [continuant]), full listings will be provided. We include only those features present (+) for each symbol in Table 2.1 as our feature set contains thirty-two (32) features in total. It should be noted that an exhaustive set of features will be utilized in the accompanying software system to maximize the descriptive power of the program.

IPA Character Symbol	**Feature Set** (positive values only)
b	[consonant] [voiced] [labial] [stop]
l	[consonant] [voiced] [coronal] [anterior] [sonorant] [approximant] [continuant] [lateral]
æ	[vowel] [voiced] [sonorant] [approximant] [continuant] [low] [front]
ŋ	[consonant] [voiced] [velar] [sonorant] [nasal]
k	[consonant] [velar]
ə	[vowel] [voiced] [sonorant] [approximant] [continuant] [mid] [central]
t	[consonant] [coronal] [anterior] [stop]

Table 2.1 Example feature sets for the symbols of the word ‘blanket’, [blæŋkɪt].

The complete list of phonetic symbols and their corresponding feature sets can be organized into a **feature matrix** which illustrates features both present and absent for a symbol. For a more comprehensive view of the feature matrix utilized in this project, see Appendix B.

2.1.2 Transcription

As already mentioned, research in phonology relies on a universal character set. To obtain an IPA string-equivalent of a sentence, **transcription** is performed on the verbal or written utterance (the source representation).

Transcription is the process of mapping speech into discrete phonetic symbols representing the individual sounds contained in the spoken utterance. Phonetic transcriptions are necessary for sound identification since there is not always a one-to-one correspondence between the language's alphabetic characters and the individual sounds of the language. Furthermore, alphabetic characters do not always associate with the same sounds across languages. A good example of this is the Latin letter 'j'. In English it could possibly be transcribed as [dʒ], in French [ʒ], in German [j], and in Spanish [x].[2] As a result, orthographic transcriptions are not very useful for linguistic research, hence the use of IPA symbols, which enable a more reliable way to identify the atomic sounds within a speech stream.

Finally, speech is not composed of sounds only. It is organized in rhythmic units. For example, stress plays a role in the composition of words or phrases. Accentuation on transcribed symbols is achieved by use of emphasis-denoting notation, or **diacritics**.

[2] Examples of these are "jump", "jamais", "jeder", and "jota" in English, French, German, and Spanish, respectively.

2.1.3 Diacritics

Diacritics are small marks added to a transcribed string that serve as a means of increasing precision. These include (but are not limited to) increased or decreased use of lip protrusion on sounds, additional nasal resonance to sounds, and so on.

An example of this uses [̪] to denote dental articulation as in the 'n' in 'tenth' yielding the transcribed string [tɛn̪θ]. Here the 'n' preceding the 'th' requires that English speakers prepare for the latter sound by placing the tongue behind the upper teeth such that the transition across sounds is made easier and less abrupt. If dental articulation were not used, the articulatory distinction between the 'n' and 'th' would be more pronounced and notably more difficult to produce.[3] This exemplifies the rationale for using diacritics to indicate phonetic attributes that are not immediately or commonly noted.

Once the transcribed string is available, syllable boundaries can be located to allow a more refined study of word composition.

2.1.4 Syllables & Syllabification

In this subsection we focus on the notion of *'syllable'*, which consists of groupings of sounds on the speech stream. For sake of clarity, we will limit our discussion to **basic syllables** only. Although, in practice, research and analysis usually focus on more complex syllable types, a discussion of the basic syllable will suffice here.

We can describe our basic syllables in a rising sonority – peak – falling sonority manner. Here, sonority refers to the relative *loudness* of sounds. We can further overlay this pattern with an *onset-nucleus-coda* sequence. Figure 2.1 illustrates that syllables themselves are not atomic, but can be further grouped into sub-syllable parts. As shown in this figure, a tree structure can represent this organization [14].

[3] One may notice by pronouncing "ten" and "tenth" that the position of the tongue while pronouncing the sound [n] is quite different. Also, one may attempt to say "tenth" without the dental articulation on the [n], noting the change.

The root of this tree is the syllable node identified with the symbol σ. A syllable, at its topmost level, is divided into an **Onset** (O) and a **Rhyme** (R). The rhyme may be further sub-divided into two constituents, commonly called the **Nucleus** (N), and the **Coda** (C). Branching is possible at any of the syllable constituent nodes to a maximum size of two (2). Syllables need not always have onsets, i.e., *eye, eat* and *ink*, nor codas, i.e., *pie* and *free*. At the leaves of the tree are the timing units, to which are attached IPA characters representing speech sounds. To understand the elaboration of a well-formed syllable, we regard the following.

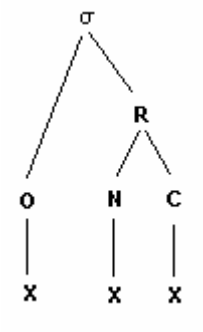


Figure 2.1 Basic Syllable Tree Structure.

The presence of one or two[4] consonants before a nucleus constitutes an onset. This indicates the rising sonority prior to the peak, where onsets exist. The mirror of an onset is a coda. This falling sonority portion of a syllable, where applicable, may signal the end of a syllable or the transition into a subsequent syllable. The peak, composed of a **monophthong or diphthong**, indicates the phonetic length (one or two **timing positions**) of the nucleus of the syllable. Nuclei may consist of a short vowel (one timing position), a long vowel (two timing positions) or a vocalic sequence (two timing positions but with two IPA symbols – the vowel and a **glide**). These nuclear forms are depicted in Figure 2.2.

[4] There are at most two consonants in the onset and one in the coda.

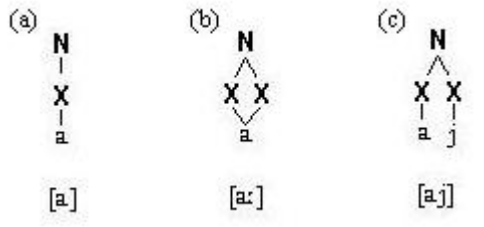


Figure 2.2 The structure of the nucleus. (a) A short vowel. (b) A long vowel, with a diacritic denoting length [:]. (c) A vocalic sequence (or heavy diphthong) with one vowel and a glide, [j].

Finally, the nucleus and coda form a phonological unit called the rhyme. The rhyme is a grouping of syllable-final sounds which is relevant for syllable well-formedness, which plays a major role for stress assignment. For the non-linguist, the rhyme is intuitively used as a basis of comparison for rhyming words. Its use therein entails structural comparisons on the number of timing positions, denoted with an 'X' in the figures [10].

Tree structures such as those shown in figure 2.1 allow us to very specifically ascertain an algorithm for locating the syllable boundaries within a word – **syllabification**. With this, we can atomize our information to a level appropriate for analysis. The [blæŋkət] and [skrijmɪŋ] examples introduced in Section 2.1.1 further exemplify word syllabification in Figure 2.3.

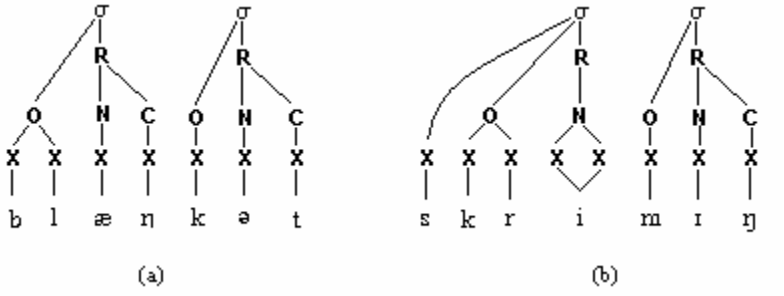


Figure 2.3 Syllabification examples. (a) Two syllables, each with and onset and coda. (b) Two syllables. Notice the binary (having two timing positions) nucleus in the first syllable.

As can be seen in this figure, an additional character can be appended to the beginning of a syllable which is not part of the onset. When a consonant cannot be accommodated in the syllable template, it is adjoined to the syllable as an appendix as seen in Figure 2.3 (b). In this case, the /s/ is appended to the left of the syllable. Appendices can also occur at the right edge of a syllable. This occurs in cases where the second position of the rhyme is already filled by another consonant. It should be noted that word-final appendices are not necessarily /s/. The transcriptions of 'caps' and 'capped' exemplify this point. These are written in IPA representation as [kæps] and [kæpt], respectively. In the latter, the coda is taken up by the /p/, causing the /t/ to be syllabified as a right edge appendix [16].

With the orthographic-to-IPA mapping as defined in Section 2.1.1 in place, word transcription performed and identification of syllable boundaries achieved, we can now begin to analyze the correlation between target speech and actual speech which are further defined and discussed in Section 2.2.2. Pertinent to the software designed for the purpose of this thesis is the need to align the syllables of the target phrase with the corresponding syllables of the actual pronunciation to systematically detect various alterations within a child's language and linguistic development over time.

2.1.5 Alignment

Alterations such as syllable changes, including modification, deletion, addition or duplication of sounds and syllables, can be detected given a method of distinguishing proper syllable positions.

To detect such processes, phonologists adopt a method of **syllable alignment** – or the synchronization of syllable positions between the target phrase and the child's actual utterance [15]. A conventional notation which will be adopted in this thesis: the use of double periods (..) to identify missing syllables.

For instance, a child attempting to say 'banana' may instead utter 'bana'. While more study is required to determine if that child is more likely removing the middle or end syllable, it is clear that one was removed. This could be denoted as either 'ba..na' or

'bana..', respectively.[5] This allows for a linear comparison on the syllables of the target ('banana') and the actual ('bana') utterances. As a result of alignment, syllable productions can be more systematically analyzed.

Looking at the English word 'potato' we can imagine a child uttering 'tato'. Clearly the target word contains three syllables, yet the actual contains only two. If alignment were not considered, then the first syllable of each word would be analyzed as if they corresponded, and similarly with the second. Consequently, analysts would compare 'po' from the target to 'ta' in the actual. While mild similarities exist (i.e. both have vowels), nothing of interest can be studied between the two. For this reason, we want to force syllables in the actual form to be comparatively aligned with their correspondents in the target, where possible. The alignment notation mentioned above provides a means of correcting this potential fault within the system and produces a syllable alignment represented by '..tato'. As a result, the proposed system will be able to perform analysis based on valid comparisons. This example is summarized in Table 2.2.

Target: potato					
Actual: tato					
Without Alignment (incorrect)			Using Alignment (correct)		
po	ta	to	po	ta	to
ta	to		..	ta	to

Table 2.2 Alignment of target and actual utterance. The table shows which syllables will be paired for analysis when alignment is absent and when it is present.

[5] It has been proposed that the first of these two is indeed more likely (Rose, PC). This is due to children's faithfulness to the sounds of a stressed syllable, as is the case with the second syllable of 'banana'.

Word-initial syllable deletion is one of the easier alignment challenges to detect. A simple comparison between target and actual phrases usually indicates which syllable is missing. Such a simple assessment is not always possible. There are four alignment challenges that must be identified: syllable reduction (as described above), vowel epenthesis, consonant epenthesis, and hiatus reduction [16].

Vowel epenthesis happens in cases where a string of sounds cannot be syllabified by the speaker but must be preserved with all its sounds in the produced form. Vowel epenthesis usually occurs to break up a sequence of consonants. Children can insert vowel sounds between consonantal sounds to recover from difficult combinations of sounds, for example in plosive + liquid clusters such as the '*bl*' in "black". Adding an unstressed vowel (such as schwa, /ə/) could result in [bəlæk] as opposed to the target [blæk] [1]. Consonantal epenthesis follows much the same pattern; an additional consonant is inserted between two consecutive vowels.

Hiatus reduction is one of the harder alignment challenges to tackle. This type of reduction occurs when the speaker cannot syllabify a vowel + vowel sequence properly. As opposed to the above, segmental preservation is not important, so that one of the vowels can be deleted from the actual vowel sequence. Also, two vowels can be reduced to a single vowel that bares some of the features of both. This event creates challenges in aligning syllables because of the ambiguity that arises when assigning the reduced syllable to a syllable in the target utterance. Figure 2.4 more concretely illustrates and summarizes these alignment issues.

Obstacle	Form	Problem
Syllable Reduction	T: CV | CV | CV A: CV | CV | ..	See beginning of this section.
Vowel Epenthesis	T: CCV A: CV | CV	There is an addition syllable in the actual form. Where does it go?
Consonant Epenthesis	T: CVVC A: CV | CVC	Similarly, we must determine where the new syllable belongs.
Hiatus Reduction	T: CV | VC A: CVC	The problem is to find reason to associate the actual syllable with one of the two target syllables.

Figure 2.4 Four alignment challenges. In the second column, "T:" and "A:" denote the target and actual syllable forms, respectively.

Despite these problematic intricacies, alignment provides a worthwhile, useful means of factoring in any alterations between the target utterance and a child's actual utterance [15].

2.2 Linguistic Analytical Software

Applications have been developed in the past twenty years that provide methods of collecting, analyzing and, to a lesser extent, sharing data. Technological advances have aided this effort and formed the basis for many powerful systems we witness today [16].

The following sections outline some of the more prominent utilities in the field of linguistics; their contributions to the computational study of data and their shortcomings are subsequently assessed.

2.2.1 The *CHILDES* Project

CHILDES, the *Child Language Data Exchange System*, was conceived in 1984 to facilitate in the creation of a shared data repository for typed, handwritten, and computerized transcripts [15]. Since then, computer development has allowed the integration of services within the *CHILDES* project that assist in audio and video linkage

between transcripts and recorded footage. *CHILDES* was been proposed to incorporate three major tools:

1. *CHAT* – a transcription and coding format meant to ease the exchange of data.
2. *CLAN* (*Computerized Language Analysis*) - an automated analysis tool for searching and analyzing data transcribed according to the *CHAT* documentation format.
3. A *minimal*, non-proprietary, database management system.

CHILDES' analytical component, *CLAN*, deals with the morpho-syntactic analysis of child language development. It implements thirty-seven analytical functions, each with between three and eleven variations, executed by the inclusion of optional parameter-flags. This functionality, however powerful, may become complex when executing some queries and also requires the researcher to have knowledge of the underlying storage mechanism. For example, to search for previously "tagged" records and find the frequency of each unique item in the result, one might run the following:

```
gem _t*CHI +d sample.cha | freq
```

Figure 2.5 illustrates an instance of *CLAN* with an open sample file, a partial list of functions, and the main command window.

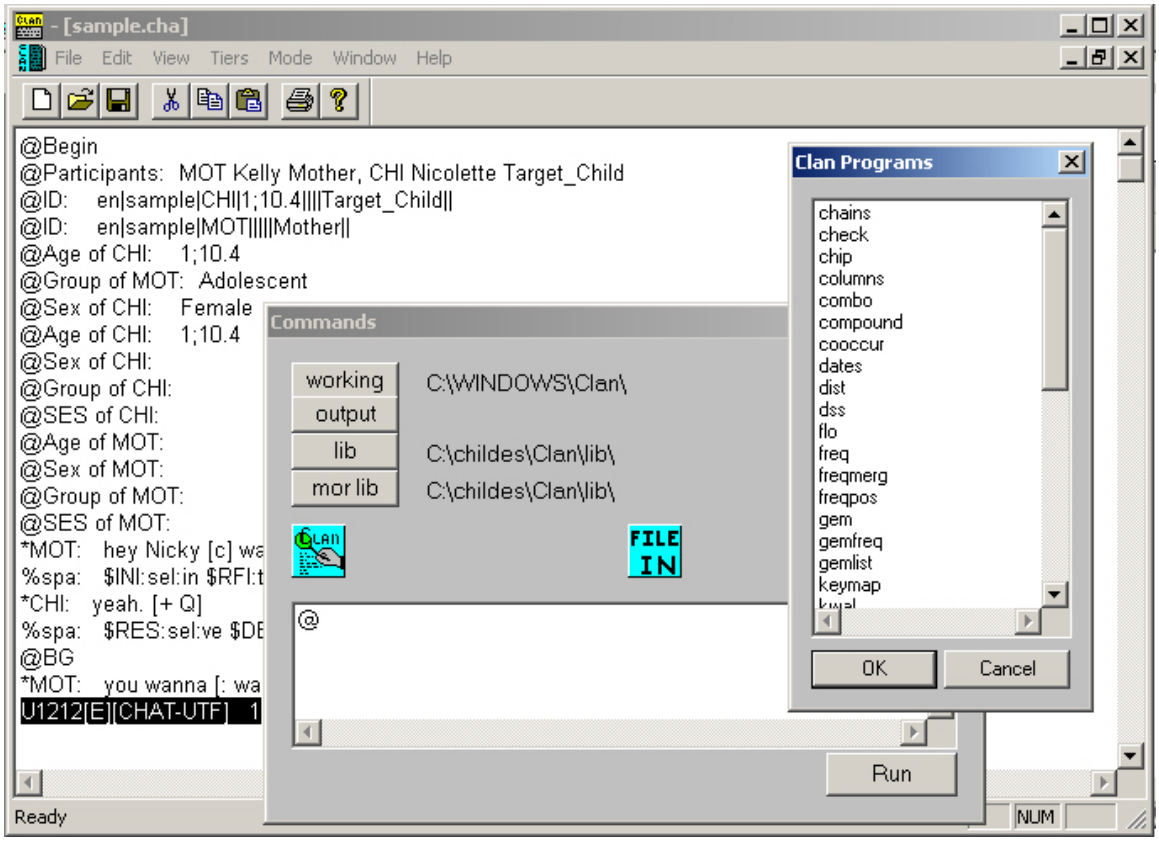


Figure 2.5 The *CLAN* application main window with open sample file, main 'Commands' frame and a partial list of functions displayed ('*CLAN* Programs' frame).

Other functionality that *CLAN* includes is its *SoundWalker* system. This utility provides a means of searching and marking audio and video data that has been linked to the transcript. Figure 2.6 depicts this interface.

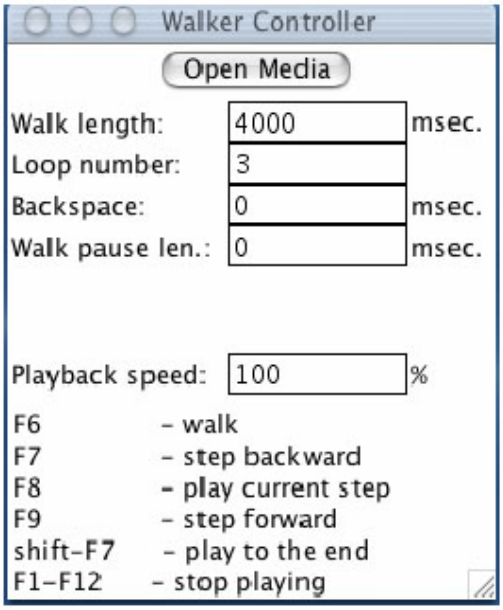


Figure 2.6 The *SoundWalker* utility in *CLAN*

Through keyboard-mapped navigation, markers can be placed and returned to at any point in the audio/video clip. Other components of the *CLAN* interface, not shown in Figure 2.6, allow access to media specific information such as media length and to wave-form visualization.

CHAT was designed to allow the extension of a fully documented data-storage format and to ease the transition into other information representations via the use of a standard format. Formats have been developed by other projects, such as *TalkBank*, to augment the *CHAT* design as needed.[6] More specific to this thesis is the need to extend this format to allow the warehousing of phonological transcription information. In Chapter 4, we propose a **tier-oriented** organization for storage and manipulation of child language-development data. This organization enforces non-destructive information-migration throughout the system and offers multiple levels of abstract analysis; this allows researches to perform analysis on a single tier without being deterred by accompanying tiers.

[6] For more on the *TalkBank* schema, see http://xml.talkbank.org.

2.2.2 *ChildPhon*

ChildPhon is a software solution for the study of child [15]. *ChildPhon* exists to fill a gap in linguistics' applications which in the past have primarily focused on the morphological and syntactic aspects of child language acquisition; *CLAN* (part of the *CHILDES* project) is an example of such an application.

Rose's *ChildPhon* is a descendant of another, obsolete application created at the Max Planck Institute of Psycholinguistics that carried the same name. The "old" *ChildPhon* however, was not released under a public license and provided only a textual interface without multimedia support and automatic analysis tools.

While *ChildPhon*'s main goal is to provide a computerized tool for analyzing phonological data from children, it is also designed to provide linguists with a user interface that provides ease of use. The current implementation of *ChildPhon* is created in the *FileMaker Pro* suite for Macintosh computers.[7] This implementation allows for a graphical representation of data as well as support for embedded multimedia files. Unfortunately *FileMaker Pro* does not currently support Unicode character fonts, preventing cross-platform compatible IPA transcriptions. As well, since *FileMaker Pro* is proprietary, software users must first purchase a license of *FileMaker Pro* in order to use *ChildPhon*.

ChildPhon's graphical interface provides linguists with an effective and simple system to aid in the study of child phonology. The application provides forms to record the following:

- Child and session specific data
- Utterance and phrase breakdown
- Transcription information
- Data Analysis

ChildPhon's forms offer linguists phonological and metadata entry fields. The metadata fields hold information such as the child's name, age, and session dates. Most importantly for phonetic research are the fields containing utterance, Adult IPA and Child

[7] *FileMaker Pro* is a work group database application (http://www.filemaker.com)

IPA data. Figure 2.7 shows *ChildPhon's* interface with these fields. The utterance field in the application consists of one or more words (possibly a sentence) obtained from a recording of the studied child. Each utterance is further sub-divided into its phrases as defined by the researcher. Phrases can consist of one or more words in the utterance (or the entire utterance itself) allowing for syllable analysis across words. Each phrase is given an adult phonetic transcription – the "ideal" case for pronunciation – and an actual phonetic transcription – what was actually produced by the studied child. The transcriptions are displayed in IPA.

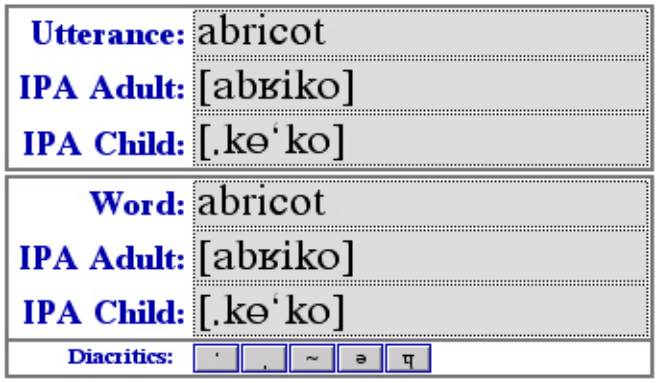


Figure 2.7 Phonological Data Entry Fields

Once the acquired information is entered into the application, *ChildPhon* is able to produce automatic processes from the data. These processes include in-depth analysis of key issues pertinent in describing phonological development such as coda sonority profiling, distinction between word medial and word-final codas, place and manner of consonant sequence articulation, and consonant harmony. Using sonority classes *ChildPhon* is able to determine the syllable boundaries of the words. Once the syllabification is completed, the application is then free to make syllable-by-syllable comparisons as shown in Figure 2.8. Currently *ChildPhon* supports the analysis of frequent processes found in syllabic contexts. These are outlined below [15]:

1. Syllable truncation (missing syllables in actual forms)
2. Syllable nucleus
    a. Long vowel shortening
    b. Short vowel lengthening
    c. Vowel epenthesis
    d. Vowel deletion
    e. Glide deletion
    f. Glide epenthesis
3. Vowel initial-words
    a. Vowel deletion
    b. Consonant epenthesis
4. Vowel-initial syllable
    a. Vowel deletion
    b. Consonant epenthesis
5. Hiatus
    a. Consonant epenthesis
    b. First/Second vowel deletion
6. Coda
    a. Coda deletion
    b. Coda epenthesis
7. s+Consonant cluster
    a. s-deletion
    b. Consonant deletion
    c. Whole cluster deletion
    d. Vowel epenthesis
    e. Vowel prothesis
8. Branching onsets
    a. First consonant deletion
    b. Second consonant deletion
    c. Vowel epenthesis
    d. Vowel prothesis
    e. Vocalization of second consonant

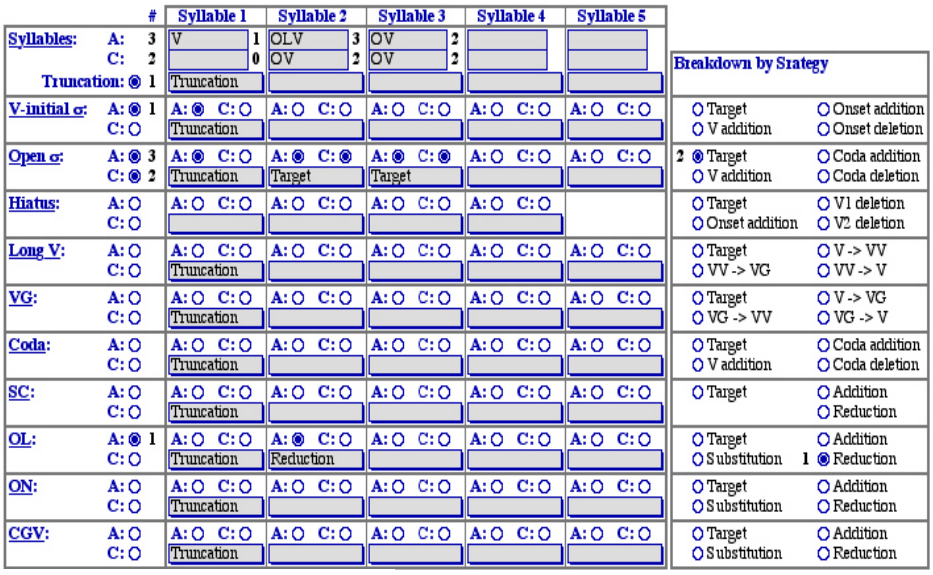


Figure 2.8 Syllable-by-syllable Comparisons

While *ChildPhon* does produce valid results, it does have some unsolvable problems in its current implementation, including:

- A Unicode implementation of the IPA character font is available which offers a more complete set of glyphs. The current version of *FileMaker* does not support Unicode.
- *FileMaker* has limited multimedia support. Support for more media formats is needed.
- Since *FileMaker* is proprietary software, the database format is closed. That is, the contents of the *ChildPhon* database cannot currently be transported easily.

Future versions of *ChildPhon* are hoped to alleviate the above issues. As well, future versions will provide data identification fields, percentages, and other mathematical functions.

2.3 XML

XML or *eXtensible Markup Language* is a subset of the *Standard Generalized Markup Language* (SGML) and is becoming widely popular for web development, application development, and data storage.[8] XML was engineered by the XML Working Group at the World Wide Web Consortium in 1996 (standardized in 1998) focusing on the following goals [20]:

1. XML shall be straightforwardly usable over the Internet.
2. XML shall support a wide variety of applications.
3. XML shall be compatible with SGML.
4. It shall be easy to write programs which process XML documents.
5. The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
6. XML documents should be human-legible and reasonably clear.
7. The XML design should be prepared quickly.
8. The design of XML shall be formal and concise.
9. XML documents shall be easy to create.
10. Terseness in XML markup is of minimal importance.

2.4.1 Document Structure

XML documents are composed of two sections: the optional prolog and the document instance. The prolog consists of the XML declaration – recommended – and the document type declaration – which is required for validation. While the prolog is optional, it is generally considered good design to include the information. The document instance is required and contains all of the logical and physical aspects of the document.

XML documents contain markup and context (data). The six most commonly used markups are shown in Table 2.3.

[8] SGML is a larger and more complex language that XML. While powerful, SGML's complexity prevented it from becoming more popular in the IT industry [9].

Type	Explanation	Examples

Element	Elements are labels that define part of an XML document. Elements may be empty or include data which can include text or other elements. Every element has a start tag, `<tagname>` and and end tag `</tagname>` with the exception of empty elements which are written as `<tagname/>`.	`<hello>hi</hello>`
		Contains text
		`<hello>`
		`<greet/>`
		Empty element
		`</hello>`
		Contains other element

Attributes	Attributes define data for elements within the start tag of that element. In general attributes help to describe the data used within the element but they do not contain the data themselves. Attributes are written as `attributname="value"`.	`<text font="arial">`
		`Some text`
		`</text>`

Parsed Entities	Parsed entities have the general form `&xxx;`. They are usually replaced with some text and are primarily used to insert a character that has a special meaning in XML.	`&amp;`
		inserts '&'
		`&lt;`
		inserts '<'

Comments	Comments are used to include optional meta-data into an XML document to help clarify sections. Comments start with `<!--` and end with `-->`. They are ignored when XML documents are processed.	`<!—`
		`This is an example of a comment`
		`-->`

Processing Instructions	Processing Instructions (PI) inform the XML processor how to interpret the next statement. XML processing instructions begin with `<?` and end with `?>`.	`<?xml version="1.0"?>`

CDATA	`CDATA` sections have the form `<![CDATA[data]]>` where data is textual information that will be interpreted as pure data. That is, the contents of CDATA sections are taken verbatim and all special characters lose their meaning.	`<![CDATA[`
		`C&P Designs`
		`]]>`
		The ampersand loses its special meaning and is displayed as typed.

Table 2.3 Common XML Markups

Prolog

The first section of the prolog is the XML declaration which may span multiple lines of an XML file and declares a document as XML. The declaration also specifies whether a document can be validated with a Document Type Definition (DTD).

The declaration consists of 5 parts:

1. Opening processing instruction - `<?xml`
2. XML version number - `version="[version#]"`
3. Encoding declaration - `encoding="[document encoding]"`
4. Standalone declaration - `standalone="[yes|no]"`
5. Closing processing instruction - `?>`

A minimal XML declaration consists of the opening processing instruction, version number, and closing processing instruction. The encoding declaration holds the character encoding type of the XML file itself. The most common types of character encoding used for XML documents are `UTF-8` and `ISO-8859-7`. The character encoding tells XML parsers how to interpret the bytes of the document based on a particular character set. When using an external DTD to validate the document instance, the standalone option should be set to "`no`". The following is and example of a valid XML version 1.0 declaration using the `UTF-8` character with a validating DTD.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
```

The document type declaration is the second section of the prolog and defines the document root and location of the validating document type definition. The document type declaration may contain the rules for the DTD thus making the XML document standalone. The general syntax of a document type declaration is as follows:

```
<!DOCTYPE RootElement (SYSTEM |PUBLIC)

       ExternalDeclarations?  [InternalDeclarations]

?>
```

Where RootElement is replaced with the root element of the XML document instance, `ExternalDeclarations` contains the location of an external DTD, and `InternalDeclarations` contain markup declarations which can be embedded into the XML document. The following example shows a valid document type declaration in which the root element of the document instance is "Employees" and the DTD for the document can be found at an external location.

```
<!DOCTYPE Employees SYSTEM

"http://somewhere.com/dtds/employees.dtd">
```

Document Instance

The document instance or document body composes most of the XML document. All persistent data contained within the document is found in the body. The first tag found in the document instance must be in accordance with the "`RootElement`" indicated by the document type declaration. The entire instance should also agree with the document type definition (if given).

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<!DOCTYPE DVDS SYSTEM
   "./dvds.dtd" ?>

<!-- document instance -->

<DVDS>
    <DVD id="1">
        <TITLE>Finding Nemo</TITLE>
        <YEAR>2003</YEAR>
        <SYNOPSIS>A fish tale.</SYNOPSIS>
    </DVD>
    <DVD id="2">
        <TITLE>Snatch</TITLE>
        <YEAR>2002</YEAR>
        <SYNOPSIS>Follow the diamond</SYNOPSIS>
    </DVD>
</DVDS>
```

2.4.2 Document Type Definition

The document type definition consists of rules that define the structure of an XML document. The rules used in the DTD are called *markup declarations*. While a full explanation of DTD's is not necessary for this thesis, we will cover the most common declarations used in a DTD [20].

Markup declarations can be an *element type*, an *attribute-list*, an *entity*, or a *notation* declaration. Explanations of these declarations are shown in the table below. They are used by XML parsers to ensure the correct implementation of an XML document.

Type	Explanation	Examples

Element Declaration	Element declarations define the name of a particular element and specify the contents contained within that element. The general format of element declarations is
	`<!ELEMENT elementName`
	`(elementContents)>`
		`<!ELEMENT`
		`people (person*)>`
		Defines an element people which may contain zero or more person elements.

Attribute-List Declaration	Attribute declarations define the attributes, their type, and their default values for a specific element. The general format of an attribute declaration is
	`<!ATTLIST elementName`
	`attName type defaultValue>`
		`<!ATTLIST person`
		`name CDATA #REQUIRED`
		`bday CDATA #REQUIRED>`
		Defines two attributes name and bday that are required for the person element, are of type CDATA, and have no default value.

Entity Declaration	Entity declarations contain reusable collections of information. The information can include text and files. The general format of an entity declaration is
	`<!ENTITY name "data">`
		`<!ENTITY`
		`gandalf`
		`"Lord of the Rings">`
		Defines the entity gandalf which references the data "Lord of the Rings".

Notation Declaration	Notation declarations contain information about external data which is passed to the processing application. The general format of a notation declaration is
	`<!NOTATION name`
	`(SYSTEM|..) "app">`
		`<!NOTATION javaprog`
		`SYSTEM "java">`
		Declares that all data marked as javaprog must be passed to the java application for processing.

Table 2.4 XML Markup Declarations

The following example displays the DTD for the DVD XML example in the previous section.

```
<?xml version="1.0"?>

<!-- root element -->
<!ELEMENT DVDS (DVD*)>

<!-- specific DVD with attribute list -->
<!ELEMENT DVD (TITLE,YEAR,SYNOPSIS)>
<!ATTLIST DVD
id CDATA     #REQUIRED>

<!-- DVD sub-elements -->
<!ELEMENT TITLE (#CDATA)>
<!ELEMENT YEAR (#CDATA)>
<!ELEMENT SYNOPSIS (#CDATA)>
```

2.4.3 XML for Data Transfer

XML is rapidly becoming the technology of choice for transferring information between applications and computers. XML is and open standard and platform independent allowing many applications to utilize the same XML format to store their data. This allows numerous applications access to the same data without the need for conversion modules. XML can also be distributed over inter-networks with great ease making it ideal for distributed applications.

As web development continues to increase, database systems must be able to communicate reliably with each other. There is a growing interest in *XML-Base Data Management* (XMLDB) for the transferal of data between these systems. As a result, many database systems can now export and import data from XML documents – thus XML is quickly becoming the communication standard for network applications [6].

2.4.4 Using XML for Linguistic Software

TalkBank is a research project encompassing many disciplines that aspires to foster fundamental research in areas of animal and human communication. It is currently funded by the National Science Foundation to researchers at Carnegie Mellon University (CMU) and the University of Pennsylvania. The project has been spear-headed by Dr. Brian MacWhinney, Department of Psychology at CMU. One of *TalkBank*'s primary objectives is to provide open source software solutions and standards for creating, sharing, and searching primary materials. *CHILDES* is currently using a multi-tiered XML schema from *TalkBank* for its database technology [18]. Currently, *CLAN*'s use of the *TalkBank* schema handles data associated with morphological and syntactic elements of linguistic research as well as multimedia information entries. The XML schema would have to be modified to include information on phonological development before it could be incorporated into *ChildPhon*.

The *Expert Advisory Group on Language Engineering Standards* (*EAGLES*) provides standards for the encoding of natural language process applications. They have created an XML-based encoding standard for linguistic corpora entitled XCES. XCES offers schema for encoding annotated data, spoken language data, and alignment data. It is an application of the Corpus Encoding Standard (CES) also managed by *EAGLES* [8].

While XCES sounds promising, it is still in beta versions undergoing constant development. Currently, *TalkBank*'s schema seems to be more stable and *CLAN* – which *ChildPhon* aspires to import data from – uses this schema. Therefore, one of the goals of *ChildPhon*'s re-engineering is to modify the *TalkBank* schema to foster compatibility between *CLAN* and *ChildPhon*.

2.4 Existing Query Languages

Only in the last 4-5 years have hierarchical query languages been proposed for navigation through semi-structured documents – indeterminate document organizations, such as XML. Emu and MATE are two such query languages that provide simple queries on sub-structures within a hierarchy, sequential queries composing these simple queries, and nested queries for complex pattern matching. These are two relatively well-known query languages to the linguistic research community.

XQuery is a product of the World Wide Web Consortium (W3C) that aims at providing a comprehensive query language specifically for XML documents. It was first introduced in April 2000.

What follows is a discussion on each of these query languages, their roots within speech annotation and their query syntax.

2.4.1 The Emu Query Language

The Emu speech database system (http://www.shlrc.mq.edu.au/emu) enables the decomposition of speech at many levels of detail and the structuring of these levels into hierarchical organizations, called **annotations**. It provides a mechanism (the Emu query language) for navigating annotations matching patterns across a single level or structures across multiple levels. Emu provides facilities for searching collections of these annotations on both sequential and hierarchical criteria [5]. Sequentially, comparisons can be made between structures immediately followed by, or preceded by, other structures. The queries can also compare two sibling nodes by looking at their size, shape, and the values at corresponding nodes in the hierarchy.

Emu annotations are arranged into levels (phonemes, syllables, words, etc.) and levels are organized into hierarchies. These hierarchies consist of further decomposition of the root-level item. Figure 2.9 gives a simple example of an Emu organization.[9] The items horizontally adjacent to the markers: *word, phonetic, tone,* etc., constitute levels.

[9] While the actual data is stored in a relational database, the queries are executed on conceptual data structures such as that shown in Figure 2.9.

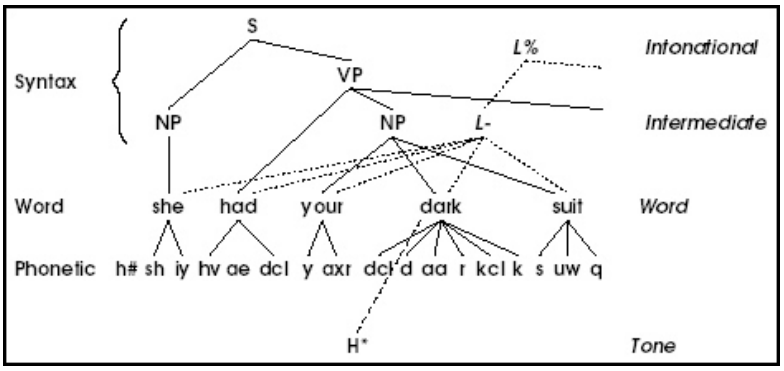


Figure 2.9 Example Emu annotation

A database of these annotations can be searched using the Emu query language. The language has primitives for simple, sequential, and hierarchical queries. A simple query follows:

```
[Phonetic=vowel -> Phonetic=stop]
```

This expression finds a vowel followed (denoted by the '->') immediately by a stop. More complex queries are built using nested simple queries. For example the following expression finds a noun-phrase (`NP`) which dominates (`^` - higher up in the hierarchy) a word "dark" that is dominated by an intermediate phrase that bears an L- tone:

```
[syntax=NP ^
     [word=dark ^
          intermediate=L- ]]
```

The Emu query lacks an explicit select clause found in Standard Query Languages (SQL). Instead, the selected material is usually the left-most structure (in this case the noun-phrase) selected in context with all relevant material to give meaning [2]. The query result is a set of these context-preserving structures.

2.4.2 The MATE Query Language

Hierarchical relationships can be represented using nested XML elements. This is the foundation governing the MATE project. The MATE project, pioneered largely by the Natural Interactive Systems Laboratory in Denmark beginning in 1998, is developing standards for annotating speech dialogue corpora [2].

The query system in MATE has a transitive dominance relation denoted by the dominance operator (^), which traverses nested structures; levels and their sub-hierarchies. For instance, to obtain the same result as the last query in the previous section, one might use the following MATE query:

```
($p phrase) ($w word) ($L intermediate);
     ($p.type = "np")
     && ($w.ortho = "dark")
     && ($L.tone = "L-")
          && ($p ^ $w)
```

Each query expression has the format of a definition and a constraint part, the definition part declares what variables are used to refer to what elements. Here, `$p` refers to a phrase element. The dominance operator indicates that the word, `$w`, will be below a phrase element in the annotation graph. Referring back to Figure 2.9, we can see that this is true. This type of declaration/dominance syntax makes for a powerful query language and eases parsing and execution. However, this syntax is rather difficult to learn and it restricts expandability.[10] Indeed, as there are a fixed set of declarative types, addition often requires much in-depth intervention.

The strategy employed by these systems is one of defining relationships between items at a specific level. These relationships include equality of items at a specified level or containment of one item within another (i.e. a root word within a complex word form). Sequential order throughout an annotation, reflected by the parent-child relationships of the hierarchy, or the identification of records given desired attributes (search criteria) is also possible.

[10] http://www.cogsci.ed.ac.uk/~dmck/MateCode/alldoc/mate/query/package-summary.html provides an application program interface for a full MATE query-interpreter.

In spite of the fact that they are powerful, these hierarchical query languages prevent the abstraction of the underlying data organization. Queries are based on the notion that the inquiring user has some knowledge about the organization of the data store, and does not base it on the theoretical structures behind it. This results in a coupling between the query and the corpus organization which is tighter than desired. An alternative may be sought in defining a language that approaches data in a more structural manner. Query languages reflecting phonological structure can provide a means of abstraction from how that language is implemented. We leave this discussion for Chapter 4 when we define the query language proposed in this thesis.

2.4.3 XQuery

For a software program that uses XML as its data storage mechanism, a query language developed solely for querying records of XML documents would seem the obvious choice.

The mission of the XML Query language (XQuery) is to provide a flexible querying facility for extracting data from XML documents. The syntax is based largely on that of XML itself [19].

The XQuery language supports operations on all the simple and complex data types of the XML Schema specification and represents collections of documents allowing queries that virtually adjoin XML documents. This is synonymous to the JOIN functionality observed in any Standard Query Language specification.[11]

XQuery provides a means of 'referencing' elements of XML documents both internally (within the same document) and externally (in separate documents). If viewed as atomic nodes, XML documents may form graphs of aggregate relations. This referencing scheme allows graphs to be traversed using like attribute values across elements.

The following is an example of an XQuery as given on the W3C website. The corresponding XML document that this query could potentially run on is not given.

[11] Examples of query languages based on SQL99 Specification, as set out by ANSI/ISO, can be found on the MySQL (http://www.mysql.com) and Oracle (http://www.oracle.com) websites.

```
<bib>
{
     for $b in doc("http://www.bn.com.bib.xml") /bib/book
     where $b/pub = "Addison-Wesley" and $b/@year > 1991
     return
          <book year = "{ $b/@year }">
               { $b/title }
          </book>
}
</bib>
```

This query would match the natural-language query "list books published by Addison-Wesley after 1991, including their year and title" [20].

XQuery was not designed with linguistic structure in mind. However, by definition, it is meant to be compatible with any XML document structure. If phonological speech is compiled and stored in a well-formed XML file, it is obvious that XQuery will be as effective on it as it would for any other document.

Chapter 3: Requirements and Analysis

Chapter 2 provided background knowledge of some of the less known topics necessary to understand how the requirements of the system will come together. Chapter three will now detail how the requirements are structured and related, as well as how these requirements elicit the necessary functionality to be incorporated into the system.

Requirements elicitation aims at defining the complete and comprehensive functionality of the system under construction. This chapter develops the specification of the software which complements this document. Specification of this type must be detailed, complete and correct for developers, yet understandable by the user.

Analysis of system requirements follows to construct a design model that software developers can unambiguously interpret. This model lays ground for the software implementation [4].

This chapter explores the problem feasibility, defines the system requirements and provides a model for the system that will be used during the software engineering stage – covered in Chapter 5. Modeling is detailed in Section 3.3 as the conceptual system model is presented based on the system requirements.

3.1 Terminology

In order to simplify our discussion and introduce consistency throughout the remainder of the chapter, the follow gives a list of terms that might otherwise be unfamiliar to the reader. These are grouped into two categories: design – dealing with the technical aspects of the system, linguistic – further linguistic detail not covered in Chapter 2.

Design

Actor

External entity that interacts with the system. Actors may be human, external system components, or hardware devices.

Conceptual Model

A visual representation of real-world entities in the domain of interest of a system. It illustrates the logical interactions of ideas, not the real interactions of entities.

Query Language

A set of predefined words and semantic rules that defines how users can make requests on a database and organize the queries results.

Report

Visual summary of the data on which a report is being generated. This can include tabular representations or graphical charts.

System Module

Autonomous unit of functionality within a system. Modules are designed such that their addition or removal do not affect the remainder of the system.

Tier

Abstract layer of information. Tiers are organized into rows, one on top of another, such that the data contained in each row is based or produced from the row below it.

Unicode Font

Font family that uses the same character code mapping on every platform. Character codes are 16 bits long allowing for 65, 536 characters in the font set. For example, the code *026E* is represented as 'ʥ' on every platform.

Use Case

Use cases are short, concise, but complete narrative descriptions that define a coherent unit of functionality provided by the system and manifested by a sequence of messages exchanged between the system and external actors

Linguistic

Actual Phrase
: Portion of speech uttered by a participant of a linguistic experiment or observation. Phrases are a sub-component of whole utterances.

Corpus
: Linguistic database. It is the repository used for making queries and performing experiments.

Segment
: A string of sounds in a media file denoted by a specific time interval relative to that media file.

Transcript
: A record of information resulting from the processing of segments. Transcripts can include orthographic and phonetic representations of a speech stream, syllable decomposition from the speech stream, and the metadata collected from segmentation.

Target Phrase
: The target pronunciation of an actual phrase. These are paired to make comparisons between idealized speech versus produced speech.

3.2 Problem Feasibility

Demand for this system is high as there is currently no linguistic software system available that incorporates all the design goals discussed within this document. *ChildPhon* is one such system that proposes to support analysis of phonological data and browsing of records within a database. Other systems discussed in Chapter 2 also provide much functionality, but lack extensibility and/or standardization.

The most important component resulting from this thesis is the query language specified later in Section 4.3. Researchers currently rely on query languages which are implemented differently in almost all linguistically-related applications. Our development of a standardized query language, which can be utilized across many applications, will lay ground for a software system that can incorporate further analytical modules for other types of linguistic research.

3.3 System Requirements

Before development can begin, we must first outline the problem statement, set goals for the system and define the transcript-document contents. We must also review system functional requirements, non-functional requirements, and necessary system attributes.

3.3.1 Goals

The leading goals are:

- To provide a computerized aid in the study of phonological acquisition. This study includes organizing and tabulating data, analyzing patterns and trends in the data, and producing reports on this study.
- To provide a multimedia environment for observational research. The experiments performed by linguists for the purpose of this software use recorded audio/video media. This is the basis for capturing analytical data.
- To build upon existing schemes for a reliable, extensible, data storage method for structural, phonological data.
- To allow the phonological corpora to be queried, browsed and exported for report generation and exchange between users, systems and applications. Multiple corpora must be tracked as additions may be made to different ones.
- To provide an extensible system that can easily and autonomously integrate new modules designed for other purposes (i.e. speech impediments, second language acquisition, dialectal variation study, etc.).

3.3.2 Functional requirements

Non-functional requirements detail the main underlying aspects of the behaviour of the system. Unlike user functionality, most non-functional activities are not directly viewed by the user, but are instead performed by the system and their results presented to an external entity of the system. This includes system functionality that facilitates data processing, formatting for data exchange, and aid in the realization of the defined use cases, described in Section 3.3.1. The functionality defined for this project is detailed in Table 3.1 below.

Function	***Details and constraints***	***Category***
Multiple Users	ChildPhon supports multiple users allowing for system controlled access.	Required
"Double Blind" Transcriptions	Each transcriber for ChildPhon is not able to access the transcriptions of the other transcribers working on the same data. This allows for the best transcriptions to be chosen and merged by the transcription merger.	Required
Multi-media Access	During segmentation, alignment of syllables, alignment of phrases and transcription, access to the recorded data must be available.	Required
Default Segment-length Constraints	To prevent overlap between two identified segments, the beginning of a segment is constrained by the position of the end of a previous segment. For any segment *i*, start time *Ts*, end time *Te* we have: $T_s(i) >= T_e(i - 1)$	Required
Recording of Metadata Information	Data external to phonological analysis is also recorded. This metadata includes the following: • Record Number • Child's name • Birth date • Session date • Child's age (computed automatically) • Session type • Utterance type • User name (user identification) • Media ID • Language • Notes	Required
Syllabification	Using sonority classes identified from the feature sets of the IPA transcriptions, the target and the actual phrases are automatically syllabified. The automatic syllabification can be edited by the researcher if desired.	Required
Target Phonetic Transcription	During the transcription phase the system will provide automatic transcriptions of the target orthographic phrase. The automatic transcription can be modified by the transcriber if desired.	Optional
Transcription Diacritics	Transcription will be supplemented with in-line diacritics. The diacritics will be available to transcribers using a system interface. Many of the diacritics are ignored in the system processes.	Required
Automatic, editable syllable alignments.	Alignment of the syllables of the target form with that of the actual form is performed immediately as a user navigates to a specific record in the corpus. Alignment provides ease of use by reducing manual input.	Required
Mapping of IPA characters to their character class.	Upon transcription, IPA symbols are associated with their character classes based on their feature set. These are consequently displayed to the user for analysis and querying.	Required
Automated syllable analysis	Structures of syllables and phonological processes are identified and reported in a visual context, including a summary of the processes across respective syllables.	Required
Dynamic numbering of syllables	Following syllabification, syllables are reported such that there is no upper bound on the number displayable.	Required
Propagation of record invalidation	If any record construct (segment, transcription, alignment, syllabification information) is modified, all subsequent information stemming from this record is invalidated.	Required

Multiple criteria queries.	Queries on the corpus can be performed on IPA symbols, features, structures within syllables, processes identified between child and adult utterances (indicated by the results of the analytical parsing).	Required
Tier-oriented data storage.	All data stored for a particular segment, including transcriptions, syllabification, alignment, and validation are stored in a non-destructive tier-based schema.	Required
Expandable utterance types.	A list of possible labels for utterance types should be expandable. The default set will include: • Imitation • Spontaneous Speech	Required
Orthographic-to-IPA character map	The system will provide a character map of IPA symbols and diacritics for phonetic transcriptions.	Optional

Table 3.1 Functional System Requirements.

3.3.3 Required System Attributes

Consequent to system design (i.e., design for portability, flexibility, and stability), certain constrains will be imposed on systems that run the accompanying software. The following table describes the essential attributes of a system required (or recommended) in order to run the accompanying software.

Attribute	***Details and Boundary Constraints***	***Category***
Java Runtime Environment	JRE provides a solution to cross-platform compatibility. Also required is the most current version of the Java Media Framework.	Required
XML Document	Data storage and exchange requires semi-structured documents to maintain phrases, transcriptions, and syllables and will use the XML formatting standard	Required
512 MB primary memory	For faster manipulation of the data repository, large portions of the database will be read into primary memory. This may require large amounts of memory.	Recommended
Unicode IPA fonts	Portability and data exchange demand a common font type to allow such exchange between systems and a consistent interface across platforms.	Required

Table 3.2 The phonological-transcription and data exchange system attributes

3.4 Requirements Analysis and Modeling

3.4.1 Defining Use Cases

The requirements listed in Section 3.2 can be logically grouped into coherent blocks called *use cases*. Use cases are used to describe the behaviour of a system under control of a user. They involve a series of events that are invoked by an actor and correspond to an operation performed by the system. Every use case contains a chronological description of its functionality, or flow-of-events, which describes the events of the use case. A *use case model* summarizes the interactions between actors and the system use cases and is shown following the list of use cases.

The following use cases describe the interactions of outside entities with the system. These use cases are a direct product of communications with the target clientele and help drive the creation of the user interface with the system. They also help in creating a full list of requirements for the system, upon which a domain model can be built.

Typical flow through the use cases is straight forward. Users are first authenticated to the system to verify the user's identity and the available roles they play in the system. Users then segment the media files using the SegmentSpeech use case. This produces a set of segments that denote the start and end time of portions of the media file that are of interest to the researcher.

Segments are then used to transcribe speech yielding a set of records in the database that show the speaker's native language transcription and the IPA transcription. These transcribes are created using the TranscribeSegments use case.

Many users can transcribe segments according to their interpretation of the data. This potentially provides a list of transcripts for a single segment. A merger may then use this set of transcripts to produce a single transcript that will be used for further processing.

Next, each transcription in the transcript is broken into phrases. Syllabification is done automatically on the phrases and the resulting syllables are then available for alignment. A validator can then verify the correct processing on the transcript up to that

point. If there are problem during processing, the validator rejects the transcript, making it available for re-preocessing. Alternatively, the validator can fix the error and verify the transcript. At this point the information is available for analysis.

Once all these steps are done, a researcher may perform queries on the database via the SelectFromCorpus use case. A research might also generate a report based on a database. The system then prompts the user to specify the information they wish to report on (by designing a query). This query is done and the report is generated.

A diagram illustrating the relationships between the various actors and the use cases is shown in Figure 3.1 below. The flow of events given above can be interpreted from top-to-bottom in Figure 3.1.

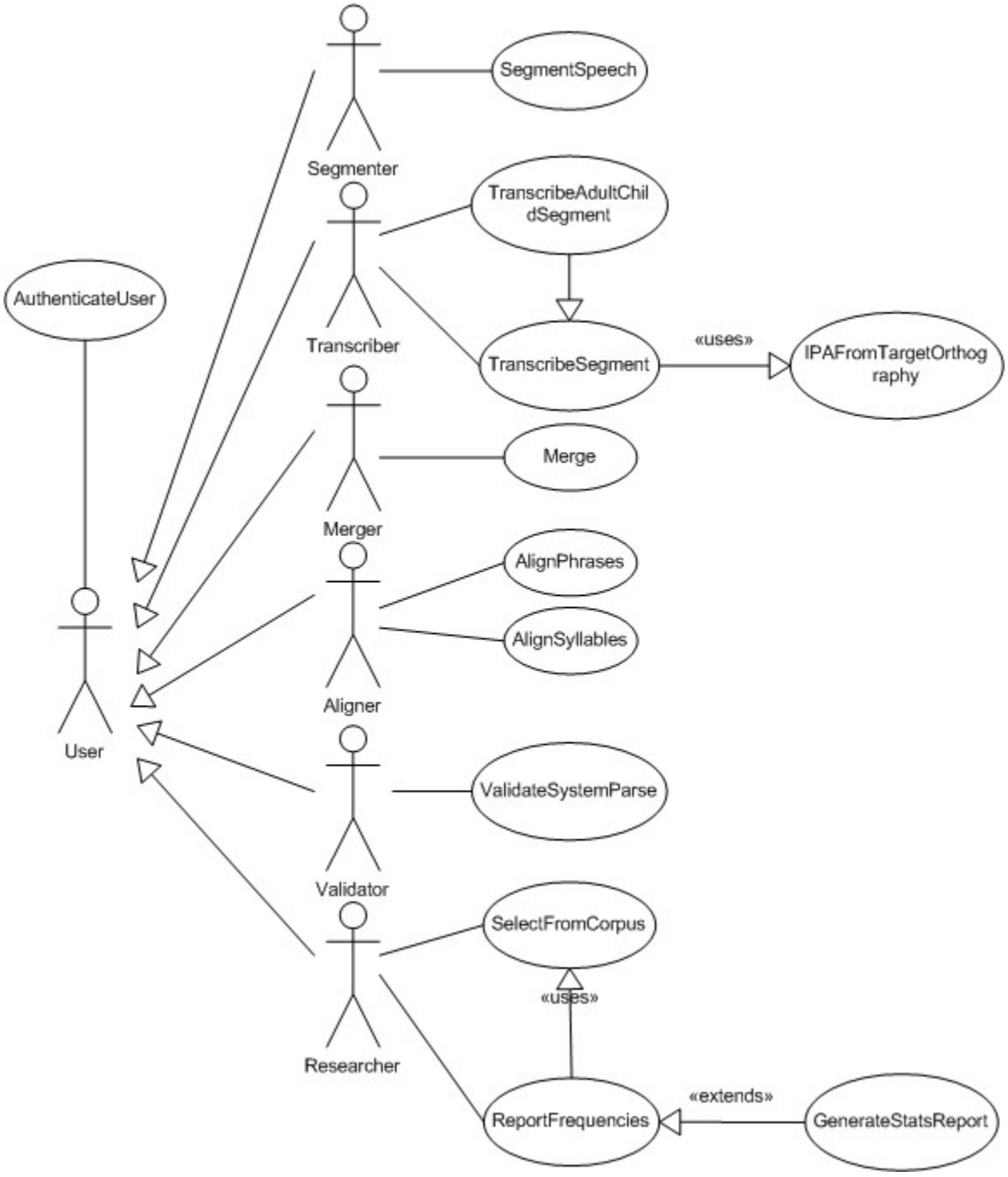


Figure 3.1 Use Case Model for the phonological transcription and data exchange system

The use cases are ordered chronologically. Many have prerequisites in the form of conditions that must be met by previous use cases. These use cases are presented in the following format:

Use Case Name	The name of the use case
Primary Actor	List of actors anticipated in the use case
Summary	An overview of the pre- and post-conditions of the use case, and its functionality
Precondition	Conditions required to be true before the use case can succeed This is where prerequisites from other use cases will be defined
Postcondition	Assumed to be true when the use case end
Flow of Events	Sequence of events
Alternative Flow	Extraordinary circumstances that do not fall in the primary flow.
Special Requirements	Requirements for the use case which are not directly related to an actor.

Use Case Name	AuthenticateUser
Primary Actor	User
Summary	All users of the system must provide, and authenticate, their identity to the system. This allows all users to gain permitted access to the system and have their data remain autonomous from other users.
Precondition	N/A
Postcondition	The general User has been authenticated on the system and their identity is known. This will persist as an implicit precondition for all of the other use cases.
Flow of Events	1. The User provides identification to the system. 2. The User provides a password to the system for authentication. 3. User confirms the validity of the entered information. 4. The System validates and accepts the User's identity.
Alternative Flow	1a. The User does not have a means of identification for this system. The User must obtain identification for the system. * steps 4a. and 4b. may be invoked repeatedly until the system accepts the identity and authentication of a requesting user. Another alternative would be to allow a predefined number of attempts and then exit the program if necessary. 4a. The system determines that there exists no User with the provided identification 1. The system notifies the user as to the probable cause of the failure. 2. The User must obtain proper identification for the system. 4b. The system determines that the password provided by the user is invalid for the given user identity. 1. The system notifies the user as to the probable cause of the failure. 2. The system requests identification and authentication from the user.
Special Requirements	N/A

Use Case Name	SegmentSpeech
Primary Actor	Segmenter
Summary	The identification of segments (time intervals) from recording media for later transcription is the goal of this use case. The Segmenter is responsible for marking the start and stop times of a segment of speech that is to be transcribed into a string of phonetic symbols. The system should provide the ability to play, rewind, and skip the media. The system should store the set of segments identified by the Segmenter.
Precondition	N/A
Postcondition	Segments have been identified for the analyzed section of the media and have been coded for production type (spontaneous speech versus imitation)
Flow of Events	1. The Segmenter enters the session information. 2. The Segmenter loads the media source that is the target of analysis. 3. Segmenter starts playing the media at the first point where no analysis (segmentation) is taken. 4. At the end of an interesting segment, the Segmenter marks and identifies the segment. 5. The system identifies a segment (interval) which has a pre-defined (default) duration and ends at the point in time the Segmenter delimits. The default interval can also be constrained by the preceding segment. 6. Steps 5 and 6 are repeated until the end of the media file is reached or the segmenter wants to end the segmentation session. 7. The system saves the identified segments and records the end of the analyzed section. 8. After segments are identified, the Segmenter can identify the break point.
Alternative Flow	* The system informs the user if there was any failure to save information on any of the segments. 6a. The Segmenter should be able to mark a section of the media as unusable. The reason for marking the section unusable should also be recorded.
Special Requirements	• Both video and audio media should be supported. • XML should be used in the encoding of the segment records. • A segment will be marked by pressing a key on the keyboards. • While some segment labels will be built in (e.g. spontaneous speech, imitation) the list of possibilities should be expandable and each expansion should be assigned a key by the system • Based on the metadata - Child's Birth-date and the Observation Date - the system will automatically compute the Child's Age. • Default segment duration can be modified. • Segment intervals cannot overlap.

Use Case Name	TranscribeSegments
Primary Actor	Transcriber
Summary	A Transcriber performs transcription on a series of segments from a recording media source.
Precondition	The segments are identified by the SegmentSpeech use case.
Postcondition	Each segment considered should have a transcription or a note indicating why the transcription was not possible.
Flow of Events	1. The Transcriber selects the sequence of segments to be analyzed. 2. Each segment is transcribed with the TranscribeAdultChildSegment use case. 3. The Transcriber can add comments about the current sequence for saving.
Alternative Flow	N/A
Special Requirements	• The system should support multiple transcribers for each segment. • Each transcriber should only have access to their transcription and no other. The requirement avoids the transcribers from being affected by the results of another transcriber. • Ability to locate media file on computer hard drive or network (in case the file gets misplaced by user). • A history of changes that can be undone will be available until navigation moves to another record. Any changes made to the record will then be saved.

Use Case Name	IPAfromTargetOrthography
Primary Actor	Transcriber
Summary	The Transcriber is responsible for entering a target phrase for transcription to phonetic symbols. The system is responsible for retrieving the phonetic transcription of the target phrase.
Precondition	The target phrase must be entered correctly (e.g. no spelling mistakes).
Postcondition	The system has responded with a suggested IPA transcription of the target phrase.
Flow of Events	1. The Transcriber enters the target phrase into the orthographic field. 2. The Transcriber is given the auto-generated IPA transcription of the target phrase. If there is more than one transcription available, the Transcriber will choose from a supplied list of transcriptions. The chosen transcription will be placed in the target transcription field.
Alternative Flow	2a. There is no transcription of the target phrase due to a spelling error. 1. The transcriber then checks to see if the target phrase was entered correctly. If so he/she re-enters the target phrase. 2. Otherwise the transcriber enters the transcription manually. 2b. There is no transcription of the target phrase due to lack of data. 1. The Transcriber enters the transcription manually 2c.There is no correct transcription of the target phrase supplied. 1. The Transcriber enters the transcription manually
Special Requirements	• The system should support editing of supplied transcriptions • The system should support automatic transcriptions of several languages. This would require the installation of the corresponding phonological database. • Automatic transcription errors will be recorded by the user and he/she will modify the transcription database.

Use Case Name	TranscribeAdultChildSegment
Primary Actor	Transcriber
Summary	The Transcriber is responsible for performing phonetic transcription on the speech identified in a segment (interval) of a recording.
Precondition	The segment should contain an utterance by a child and a target version of that utterance.
Postcondition	The system has stored a transcript of the segment with: • The natural language written version of the utterance • The phonetic transcript of the child's utterance • Segments with no transcription should also be stored with a reason of the inability to create a transcription.
Flow of Events	1. The Transcriber plays the segment. The segment can be replayed as many times as required by the Transcriber. Intervals of the segment can also be played. 2. The transcriber records the written language version of the target's utterance. 3. The phonetic version of the target (idealized) utterance is transcribed using IPA phonetic characters. 4. The phonetic version of the child's utterance is transcribed using IPA phonetic characters.
Alternative Flow	* Failure to save the transcription record should be logged. 2a. The transcriber can mark the segment as unusable with a reason. The follow steps are then skipped.
Special Requirements	• The system should support transcription from different recording media. • The system should support the IPA character set. • A visual chart of the character set can be displayed, from which the IPA character can be selected. • If qualified, the transcriber can modify segmentation of the utterances. • A history of changes that can be undone will be available until navigation moves to another record. Any changes made to the record will then be saved.

Use Case Name	MergeTranscriptionsSegment
Primary Actor	Merger
Summary	The Merger is responsible for merging two or more transcriptions of a particular segment. Cases where the transcriptions are identical can simply be marked as merged. The merger has to reconcile any differences between two or more transcriptions.
Precondition	N/A
Postcondition	The transcription of a segment has been validated and the transcription is marked as valid. Alternatively, the transcriptions can not be reconciled, and this segment is marked for future review.
Flow of Events	1. The system displays the set of transcriptions for a particular segment. 2. If all transcriptions of the segment are identical, the transcription is considered correct and automatically chosen by the system. 3. The Merger can play the recording of the segment as many times as required to confirm or reconcile the transcription. Different intervals can also be played. 4. The Merger decides upon a valid transcription, provides the valid transcription, and marks the segment as valid. The original transcriptions are retained for further study.
Alternative Flow	N/A
Special Requirements	• There should be a system option to enable or disable the automatic checking. • A history of changes that can be undone will be available until navigation moves to another record. Any changes made to the record will then be saved.

Use Case Name	AlignPhrases
Primary Actor	Aligner
Summary	The Aligner evaluates utterance and transcription to determine the phrase boundaries such that the orthographic, IPA Target, and IPA Actual transcriptions are aligned consistently. The Aligner confirms identification and adds results to the system for analysis.
Precondition	The utterances produced have been transcribed in orthographic and phonetic symbols for the Target forms and in phonetic transcriptions for the Actual forms.
Postcondition	Phrase-boundaries have been identified, or notes have been made as to why alignment was not possible, and results have been stored for analysis.
Flow of Events	1. The Aligner selects the orthographic/transcription pair for use. 2. Aligner identifies boundaries in transcription to provide consistent alignment with orthographic and transcribed utterance representations. 3. Aligner confirms alignment for utterance group and indicates completion for this pair. 4. Alignment mapping information is stored by system.
Alternative Flow	*The system informs the user if there was any failure to save information on any of the phrases 2a.Orthographic/transcribed pair may not have a valid word-boundary alignment. Aligner notes inability to record word-boundary alignment.
Special Requirements	• The system could offer automatic alignment for a default value. This automatically generated result may be edited. • Phrase identification may involve 'null' mappings from orthographic/transcription to the phrase alignment result. • A history of changes that can be undone will be available until navigation moves to another record. Any changes made to the record will then be saved.

Use Case Name	AlignSyllables
Primary Actor	Aligner
Summary	The alignment of syllables in the Target/Actual IPA transcription pair. The Aligner evaluates utterance and transcription to determine the boundaries between syllables such that both forms are aligned consistently. The Aligner confirms identification and adds results to the system for analysis.
Precondition	Phonetically clustered phrases have been identified and aligned (see AlignPhrases use case).
Postcondition	Syllable boundaries have been validated/identified, or notes have been made as to why alignment was not possible. The results have been stored for analysis.
Flow of Events	1. The Aligner selects the Target/Actual transcription pair for use. 2. The system provides a syllabification for the Target/Actual transcription pair. 3. The system provides a suggested alignment. 4. The Aligner validates the system's suggestion. 5. Alignment mapping information is stored by system and made visible for analysis.
Alternative Flow	* The system informs the user if there was any failure to save information on any of the syllables (syllable pairs). 4a. The system guess is incorrect. The Aligner alters the alignment and confirms manual-alignment's correctness. 4b. Syllable alignment is not possible. The Aligner makes a note as to why the alignment was not possible.
Special Requirements	• System could offer automatic alignment for a default value. This automatically generated result may be edited. • Syllable identification may involve 'null' mappings from Target to Actual result. (i.e. in the case of inserting '..' to denote a missing syllable.) • A history of changes that can be undone will be available until navigation moves to another record. Any changes made to the record will then be saved.

Use Case Name	ValidateSystemParses
Primary Actor	Validator
Summary	A Data Validator manually performs verifications of the system's automatic parses and comparisons and validates (confirms) these computations.
Precondition	The words contained in the corpus have been identified.
Postcondition	The parses and comparisons performed by the system are verified by the Validator.
Flow of Events	1. The Validator examines the system's parses and comparisons. 2. The Validator confirms correct computations. 3. The system records the confirmation.
Alternative Flow	* The parses and/or comparisons are not valid. 2a.The Validator modifies the result accordingly.
Special Requirements	N/A

Use Case Name	SelectFromCorpus
Primary Actor	Researcher
Summary	A Researcher performs queries from the transcribed data records, associated metadata, and/or the results of the system's automatic linguistic structure identification, or data comparisons.
Precondition	The atomic units, transcribed syllables, in the corpus have been identified.
Postcondition	The system has extracted the data based on the search criteria defined by the Researcher.
Flow of Events	1. The Researcher selects the search criteria from any of the available interfaces including transcription metadata, the results of the system's automatic linguistic structures, or data comparisons. 2. The system performs the query based on the search criteria. 3. The system displays the results of the query by providing the subset of the data which fit the search criteria. 4. The Researcher browses through the records provided by the query.
Alternative Flow	4a. The query was incorrectly set up by the Researcher 1. Researcher indicates need to re-select criteria for another query. 2. The Researcher is presented with an interface allowing a new query. 3. The SelectFromCorpus use case is re-invoked by the system. 4b. The query is not specific enough 1. The Researcher refines the query based on the results obtained from the preceding query.
Special Requirements	• Possible criteria include character string in transcriptions, information in the metadata fields, results obtained from the system's automatic identification of linguistic structures in the query, and results obtained from the system's automatic comparisons between TargetIPA and ActualIPA fields. • Set operations (such as 'AND' and 'OR') enable intersection and union of the criteria in a multiple-criterion query.

Use Case Name	ReportFrequencyCounts
Primary Actor	Researcher
Summary	Reporting the frequency of phonology queries provides data for research in phonology. The system counts the number of occurrences of a particular phonology query from a set of (possible confirmed) records selected by a query.
Precondition	A valid corpus of transcription records exist.
Postcondition	The system reports the counts for the query as a function of the total size of the corpus or the size of a defined subset of the corpus (that subset which contains records relevant to the query in question).
Flow of Events	1. The researcher selects the source corpus. 2. The researcher selects the set of transcript records of interest with a query using the SelectFromCorpus use case. 3. Using the selected records, a second query using the SelectFromCorpus use case is used to identify a subset of the first query. 4. The system reports the counts (ratios) of the two queries.
Alternative Flow	2a. If the initial query fails to find any matching records, then the system stops with a message indicating that no records were found. 3a. A second query is not required. The Researcher may be performing a more comprehensive study.
Special Requirements	N/A

Use Case Name	GenerateStatsReport
Primary Actor	Researcher
Summary	Once the values are selected by the system after a query is performed by the researcher, the system generates a stats report. These reports typically consist of proportions of target-like realisations of a given phonological structure at a given point in time or across several points in time (this produces a learning curve). They may also be a breakdown of phonological processes affecting a given phonological structure at a given point in time or across several points in time.
Precondition	A list of values is identified by the system
Postcondition	The system has generated a stats report based on the results of the ReportFrequencyCounts use-case.
Flow of Events	1. The researcher confirms that the last query performed is the one needed. 2. Researcher indicates need to generate a statistical report on the results of the frequency counts. 3. The system generates the stats report and presents the results to the Researcher.
Alternative Flow	1a. The query results currently being presented are not for the desired query. The Researcher must return to invoke ReportFrequencyCounts use case.
Special Requirements	• The stats report is either a list of numerical values or a graph plotting these values [textual vs. visual]. In the event of a textual report, the numerical values are generated as a tab- delimited text file. • Potential types of graphs: Scatter graph, Line graph, bar chart, pie chart, area graph. • The graphs generated by the system can be saved as a tiff or jpg file so it can be used in text processing or presentation software. • Stats reports can be based on a single child/session, single child/sessions or multiple children/sessions. Indeed, they may be based on a single child or multiple children pertaining to any of the particular processes.

3.4.2 Defining the Conceptual Model

From the use cases we can obtain the conceptual model – sometimes called the domain model. The process of discovering conceptual classes within the domain model is a vital step in the analysis or the system requirements. Within our conceptual class diagram we have a static structure of concepts or domain objects, associations, which in turn may have multiplicities, and attributes defined for a concept. Figure 3.2 helps to illustrate these examples.

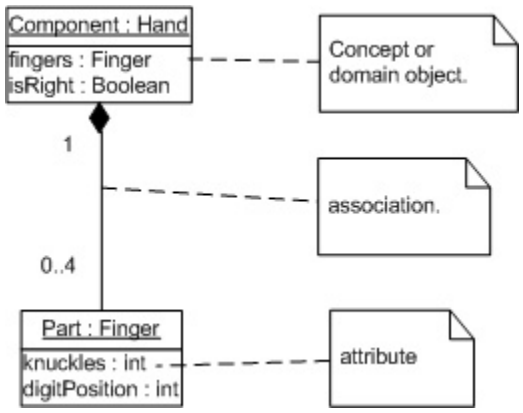


Figure 3.2 Example conceptual model components.

Unlike the use case model which concentrates on the sequential or chronological flow of events and user interactions of the proposed system, the conceptual model is an object-oriented analytical tool. It relates the fundamental interactions between the high-level concepts in the problem domain. Concepts denoted within a domain model need not directly correspond to components of the software, nor of the semantics of the system design. However some concepts may have a realization within the development of the system through assignment of roles and responsibilities [13].

Figure 3.3 models the concepts of phonological transcription, structural queries on the phonological data, analytical research and data exchange for this thesis and the ancillary software system.

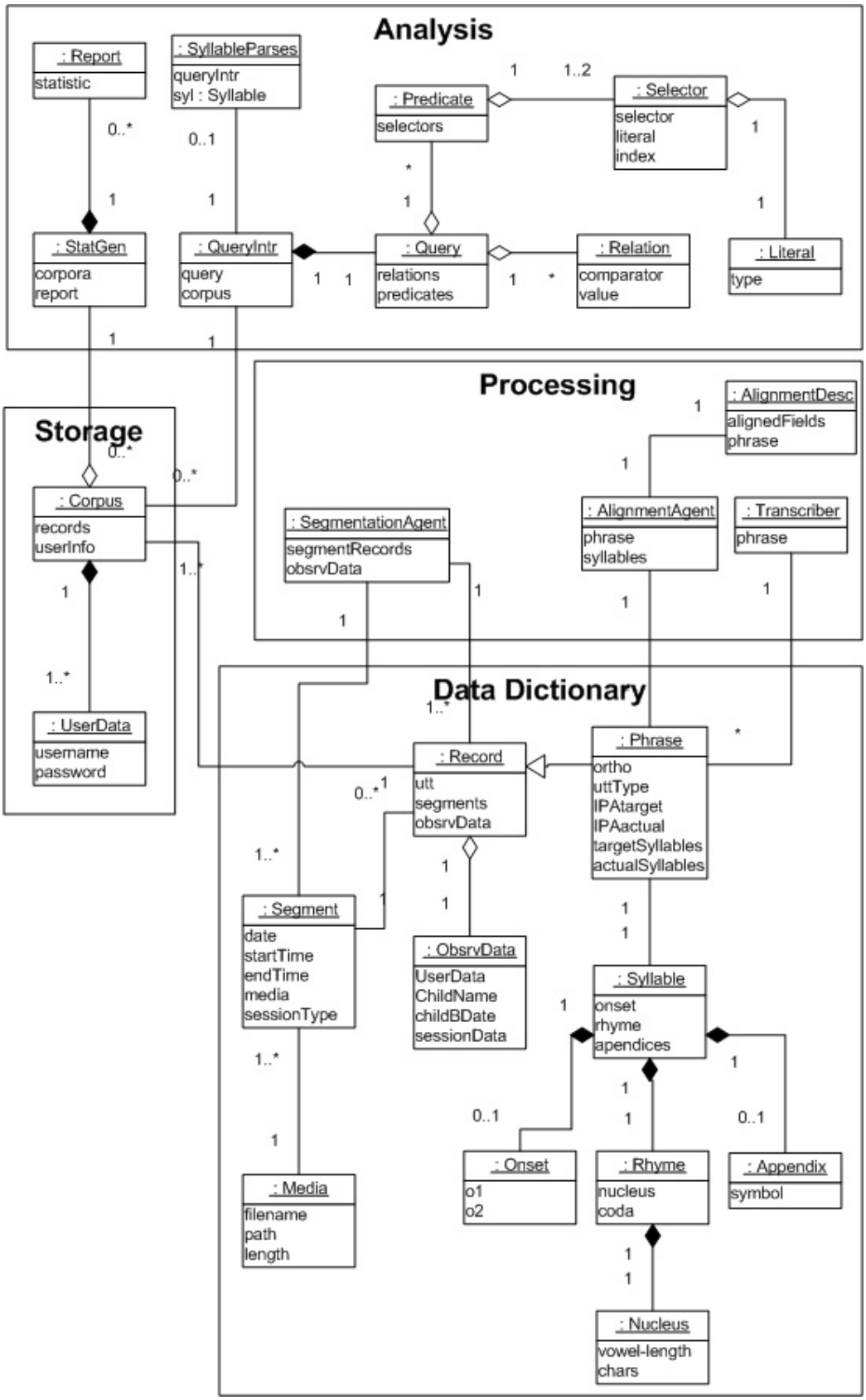


Figure 3.3 Phonological Transcription and Analysis System conceptual model

The following details the roles or purposes of these objects and are grouped by the four major divisions in the diagram:

Data Dictionary

Corpus

The organization of the linguistic database. It consists of a set of experiment Records and UserData. A Corpus can be the original database, or the result of a query.

UserData

Information on the human users of the system and the participants involved with linguistic study experiments.

Storage

Record

Root element of a database entry. It contains the orthographic utterance for a participant of an experiment. A Record also stores the observational data and a set of Segments that are derived from multimedia files.

Phrase

Specialization of a database entry. It is a Record but also has information on the IPA transcriptions of target and actual speech. Phrases have a set of Syllables that store the information about elementary units of speech.

ObsrvData

Meta data relating to an experiment. This may include, date, child's name, child's birthdate, researchers identity.

Segment

Information on the portion of a Media file that a Record comes from. It delimits the time interval for the associated Media.

Media

The audio or video source for an experiment.

Syllable

Elementary units of phonological experiments. Can be decomposed into atomic units called *constituents*.[12]

[12] Constituents were described in Section 2.1.

Onset

Constituent of a Syllable. It consists of the consonants that precede the Nucleus.

Rhyme

Constituent of a Syllable. It is the nucleus together with a consonant following the Nucleus.

Nucleus

Constituent of a Syllable. It is the vowel of a syllable. The Nucleus constituent is an obligatory constituent.

Appendix

Non-*core* constituent of a Syllable. An appendix is the consonants that either follow or precede the core Syllable (the constituents mentioned above).

Processing

Transcriber

Controls the creation of transcripts from Phrases.

SegmentationAgent

Mediates creation of Records from Segments.

AlignmentAgent

Uses a set of Phrases to create an AlignmentDesc. This provides a means of preparing information for analysis.

AlignmentDesc

Describes how syllables are paired across Phrases.

Analysis

Report

Summary of the results of a query. This can be presented in either tabular of graphical format.

StatGen

Used to generate statistics from a Corpus. It uses the QueryIntr to generate a report from a given Corpus.

QueryIntr

A Query Interpreter. It uses Query trees to match patterns in a Corpus to a request from a user.

Query

Request that can be made on a database. It consists of set of Relations and Predicates.

Relation

Used to compare two elements. Relations have comparators and values. Comparators can be <, >, =, ≠, ∩, etc.

Predicate

Specialized boolean functions. For example we might use: `branching( onset(Syllable) )` – this could be *true* or *false*.

Selector

Functions to select parts of complex structures. For example, we might have: `onset( Syllable )` – this would select `A`.

Literal

Primitive query element type. Literals can be a character (i.e. "a" or "æ"), string of characters (i.e. "aligner") or an integer.

SyllableParses

While queries are performed by users, the system will have a set of predefined, commonly used, queries that parse each Record of the Corpus for quick analysis on some of the common speech patterns when navigating through the Records.

Chapter 4: System Design

4.1 Terminology

Similar to Chapter 3, we first present some terms that will help in the remainder of the chapter.

Grammar
: A set of rules, called productions, used to define the syntax of a language. The grammar has a set of terminal and non-terminal symbols and a starting symbol. Non-terminals define the source or left-hand-side of a production [21]. A grammar will be shown in Section 4.2.9.

Predicate
: Functions which return a Boolean (TRUE or FALSE) result.

Selector
: Functions that extract data from a complex structure. Selectors can be nested to navigate a hierarchical structure.

Syllabification
: A process for identifying the syllable boundaries in a speech stream.

Syllable Alignment
: The process of arranging syllables across two or more phrases such that all syllable positions are synchronized with that position in another phrase.

Variable Quantification
: Putting variables of a query string into context. Quantification defines scope – or the part of a query where a variable has meaning.

4.2 Phonological Algorithms

From Figure 3.3 we can see a specific relationship exists between the constituents of a syllable. This is shown in the lower right-hand corner of the *Data Dictionary* box. System design will first look at modeling this structure which is completely dependent on the speech stream, thus each structure may be different.

To obtain this structure from an IPA string, we will introduce an algorithm called syllabification. Once syllables are created from the target and actual phrases (these terms are detailed in Section 3.1), we will need a means of pairing the corresponding syllables between these two sets. For this we define an algorithm called Alignment. The following two sub-sections give these algorithms.

4.2.1 Syllabification

The process of syllabification was introduced in Section 2.1.4. For the purpose of implementation, a concrete algorithm must be designed to decompose syllables into their sub-elements as described in Section 2.1.4.

Syllabification is language specific. There are certain constraints imposed by some languages that change the way words, phrases, and sentences can be syllabified. To provide an effective and accurate algorithm, these constraints, represented as parameters to the algorithm, must be taken into account. These affect how characters are assigned to particular syllable constituents. In the algorithm defined at the end of this section, we use the following language-specific parameter:

- LANG_COMPLEX_ONSETS – if the language allows complex (binary) onsets. This is necessary since some languages allow only singleton onsets.

While much more sophisticated syllabification can be designed for linguistic study, these parameters should be constrictive enough.[13] The following algorithm illustrates the syllabification process. Note that a *syllabic element*, in line 5, is either an IPA vowel character or a syllabic consonant.

Syllabify(P)

```
1  S ← [ ]
2  c ← 0
3  o ← length[P] + 1
4  for m ← length[P] to 1
5         do
6         if syllabic(P[m])
7                   c ← m
8                   N.ipa ← P[m]
9         if P[m + 1] == 'ː' or lax(P[m + 1])
10                  N.long = true
11                  c ← m + 1
12        else if glide(P[m + 1])
13                  N.ipa ← concat(N.ipa, P[m + 1])
14                  c ← m + 1
15        R.n ← N
16        c ← c + 1
17        if c < o and consonant(P[c])
18                  R.c ← P[c]
19        Syl.r ← R
20        if consonant(P[m - 1])
21                  o ← m – 1
22                  O.ipa ← P[o]
23                  Syl.o ← O
24        if consonant(P[o - 1]) and sonority(P[o]) – sonority(P[o - 1]) ≥ 1
                    and P[o - 1] != 's' and LANG_COMPLEX_ONSETS
25                  o ← o – 1
26                  O.ipa ← concat(P[o], O.ipa)
27                  InsertAtFront(S, Syl)
28        for i ← length[P] to 1
29                  do
30                  if unsyllabified(P[i])
31                            if i is word-final
32                                      append P[i] to last Syl in S
33                            else
34                                      append P[i] to Syl immediately following P[i]
35                            end if
36        end for
37 end for
38 return S
```

[13] Children will not necessarily break-down speech into syllables that conform to the "rules" of the target language. There are other restrictions in the onset, for example, but we cannot impose these restrictions on child language acquisition.

This algorithm works from the end of the line back to avoid taking characters as codas when they belong to onsets of the following syllable. The structure S is an array that holds the Syllables once they are constructed. When a new syllable is discovered in the speech stream it is inserted at the front of *S* to conserve the ordering of the syllables (since they are discovered in reverse order).

The variables *c* and *o* are used for this purpose. When each syllable has been composed, '*o*' marks the left-most IPA character of that syllable. When the next syllabic element (closest one to the left of this syllable) is found, *o* will be interpreted to mark the syllable boundary such that *c* will not be able to consume a character that has been syllabified.

4.2.2 Alignment

Once syllabification is complete, it is necessary to align the syllables for the target and actual phrases so that accurate syllable analysis can take place. To perform the alignment we use a form of semi-global alignment extracted from algorithms for computational biology [12]. The algorithm *align* is comparable to the *Longest Common Subsequence* algorithm and returns an alignment matrix [7]. From the given alignment matrix, we are able to use a traceback function to determine the computed alignment. *Align* takes two parameters, *X* and *Y*, which are sequences of Feature Sets. The algorithm is outlined below.[14]

[14] The alignment algorithm can be implemented using a recursive function. However, for simplicity the more general, looped version is shown.

```
Align( X,Y )
1  m ← length(X)
2  n ← length(Y)
3  indelCost <- -1
4  for i ← 1 to m
5          do c[0,i] ← 0
6  for j <- 1 to n
7          do c[j,0] ← j * indelCost
8  for i ← 1 to n
9          do for j ← 1 to m
10                 do
11                 x ← c[i,j] + sim(X[j], Y[i])
12                 y ← c[i-1,j] + sim(X[j], '-')
13                 z ← c[i,j-1] + sim('-', Y[i])
14                 if x >= y and x >= z
15                         c[i,j] ← x
16                         b[i,j] ← '↖'
17                 else if y > x and y >= z
18                         c[i,j] ← y
19                         b[i,j] ← '↑'
20                 else if z > x and z > y
21                         c[i,j] ← z
22                         b[i,j] ← '←'
23                 end if
24         end for
25 end for
26 return c and b
```

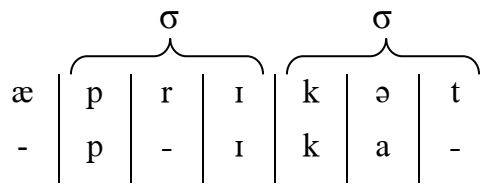
Within the *align* algorithm, we use a function *sim* which computes the similarity between two given Feature Sets. The similarity is based on the intersection of the two given Feature Sets. This function is extremely important to the function of *align*. Modifications to this function will allow for refinement of the alignment results. For an example we give a very simple similarity function in which indels have a value of -1, the feature 'Vowel' has a value of 4, and all other features have a value of 1. Our reasoning for giving vowels a high alignment value is so that syllables will be attracted to each other by their nuclei.

```
sim(X, Y)
1  if X = '-' OR Y = '-'
2          return -1
3  end if
4  z ← X INTERSECT Y
5  total ← 0
6  for each p in z
7          do
8          if p = 'Vowel'
9                  total ← total + 4
10         else
11                 total ← total + 1
12         end if
13 end for
14 return total
```

The following table shows the result of the *align* function given the target phrase "æprɪkət" and the actual phrase "pɪka". The variables '**i**' and '**j**' refer to those variables in the above algorithm.

	i	æ		p		r		ɪ		k		ə		t	
j	0		0		0		0		0		0		0		0
p	-1	↖	0	↖	3	←	2	←	1	↖	2	←	1	↖	2
ɪ	-2	↖	4	←	3	↖	3	↖	8	←	7	↖	6	←	5
k	-3	↑	3	↖	5	↖	4	↑	7	↖	11	←	10	←	9
a	-4	↖	2	↑	4	↖	5	↖	8	↑	10	↖	15	←	14

Once the alignment matrices are complete, we can use a trace to find the alignment. The trace begins with the highest value in the last row of the table. We then follow the arrows until we reach the first row of the table. The trace for the above example is shown in dark blue. Using this trace we are given the correct alignment shown below.



Now the second and third syllables of the target phrase are aligned with the first and second syllables of the actual phrase.

4.3 Query Language Specification

The following sections describe the *predicates* and *selector* functions available in the core query language as well as sample queries defining searches on the syllable processes defined in Section 2.2.2. The language will use a combination of prefix-notated and infix-notated functions.

4.3.1 Searchable Context

The table below displays the criteria used in searching the main corpus.

Type	*Fields*
Observable Data	1. Participant's name 2. Birth date 3. Session date 4. Participant's age (computed automatically) 5. Session type 6. Utterance type 7. Segmenter's name (user identification) 8. Merger's name 9. Aligner's name 10. Validator's name 11. Media ID 12. Language
Syllable Structures	13. Onset 14. Rhyme 15. Nucleus 16. Coda 17. Appendix
Phrase	18. Orthographic text 19. IPA Target 20. IPA Participant
Syllable Process	As listed in Section 2.2.1.

Table 4.1 Query Context

Each query can iterate through all the records in the corpus. Each record holds the contexts described in the table above and selector functions are used to extract the relevant data. Records have the following format:

Record

1. Observable data
   a) Participant's name: string
   b) Birth date: date
   c) Session date: date
   d) Participant's age (computed automatically): integer
   e) Session type: string
   f) Utterance type: string
   1. Segmenter's name (user identification): string
   2. Merger's name: string
   3. Aligner's name: string
   4. Validator's name: string
   a) Media ID: string
   b) Language: string
2. Utterance (the full sentence in orthographic representation): string
3. Target Phrase: Phrase
   a) IPA Transcription: string
   b) Orthographic Transcription: string
   c) Syllable1: Syllable
      - Onset: Onset
      - Nucleus: Nucleus
      - Rhyme: Rhyme
      - Coda: Coda
      - Appendix: Appendix
   d) Syllable 2: Syllable
      - ... (for as many syllables as necessary)
4. Participant Phrase: Phrase
   a) IPA Transcription: string
   b) Orthographic Transcription: string
   c) Syllable1: Syllable
      - Onset: Onset
      - Nucleus: Nucleus
      - Rhyme: Rhyme
      - Coda: Coda
      - Appendix: Appendix
   d) Syllable 2: Syllable
      - ... (for as many syllables as necessary)

4.3.2 Data Types

During query evaluation, many data types are evaluated and compared. These data types may consist of basic structures such as strings, integers and dates or more complex structures such as syllable constituents.

When searching observational (meta) data, the primary types used include strings: subject's name, language, integers: subject's age, and dates: date of recording, and date of birth of subject. Queries involving syllabic data will use complex data types such as *Feature Sets* (defined below), Syllable Parts, and Phrases. The associations of these data types are outlined in the figure below.

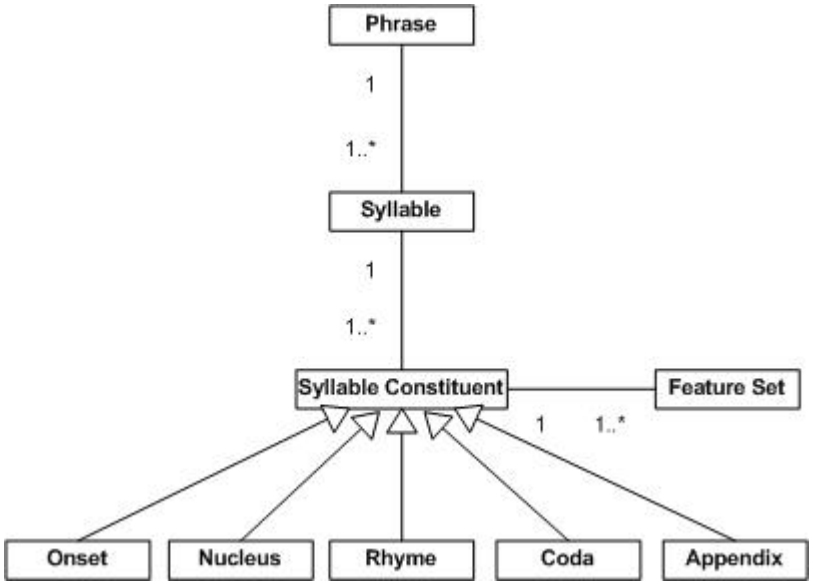


Figure 4.1: Syllable Query Data Types

4.3.3 Predefined Query Functions

Here we will list the predefined functions available in the query language.

Selector Functions

Function Name: actualPhrase
Notation: Prefix
Parameters: Record
Returns: Phrase
Summary: This function will return the actual Phrase of the given Record.

Function Name: birthdate
Notation: Prefix
Parameters: Observable Data
Returns: Date
Summary: This function will return the date of birth of the subject.

Function Name: coda
Notation: Prefix
Parameters: Syllable
Returns: Coda
Summary: This function will return the Coda structure of the given Syllable.

Function Name: IPATranscription
Notation: Prefix
Parameters: Phrase
Returns: String
Summary: This function will return the unicode string representing the IPA transcription of the given phrase.

Function Name: language
Notation: Prefix
Parameters: Observable Data
Returns: String
Summary: This function will return the language of the session.

Function Name: lappend
Notation: Prefix
Parameters: Syllable
Returns: Feature Set
Summary: This function will return the Feature Set at the initial timing position of a Syllable.

Function Name: nextSyllable
Notation: Prefix
Parameters: Phrase, Syllable
Returns: Syllable
Summary: This function will return the next Syllable in the specified Phrase using the given Syllable as a reference.

Function Name: nucleus
Notation: Prefix
Parameters: Syllable
Returns: Syllable Nucleus
Summary: This function will return the Syllable Nucleus structure contained within the given syllable.

Function Name: observableData
Notation: Prefix
Parameters: Record
Returns: Observable Data
Summary: This function will return the Observable Data structure contained within the given Record.

Function Name: onset
Notation: Prefix
Parameters: Syllable
Returns: Onset
Summary: This function will return the Onset structure contained within the given syllable.

Function Name: orthoPhrase
Notation: Prefix
Parameters: Phrase
Returns: String
Summary: This function will return the string representing the orthographic transcription of the given phrase.

Function Name: orthoUtterance
Notation: Prefix
Parameters: Phrase
Returns: String
Summary: This function will return the string representing the orthographic transcription of a record's utterance.

Function Name: pos
Notation: Prefix
Parameters: Any Structure
Returns: Feature Set, integer
Summary: This function will return the child structure at position i in the given Structure.

Function Name: previousSyllable
Notation: Prefix
Parameters: Phrase, Syllable
Returns: Syllable
Summary: This function will return the previous Syllable in the specified Phrase using the given Syllable as a reference.

Function Name: rappend
Notation: Prefix
Parameters: Syllable
Returns: Feature Set
Summary: This function will return the Feature Set at timing which corresponds to the final appendix of a Syllable.

Function Name: rhyme
Notation: Prefix
Parameters: Syllable
Returns: Rhyme
Summary: This function will return the Rhyme structure contained within the given syllable.

Function Name: sessionDate
Notation: Prefix
Parameters: Observable Data
Returns: Date
Summary: This function will return the date of the session with the subject.

Function Name: sessionType
Notation: Prefix
Parameters: Observable Data
Returns: String
Summary: This function will return the type of the session with the subject.

Function Name: subjectAge
Notation: Prefix
Parameters: Observable Data
Returns: Integer
Summary: This function will return the age of the subject at the time of the session.
Function Name: subjectName
Notation: Prefix
Parameters: Observable Data
Returns: String
Summary: This function will return the name of the subject.

Function Name: targetPhrase
Notation: Prefix
Parameters: Record
Returns: Phrase
Summary: This function will return the target Phrase of the given Record.

Function Name: utteranceType
Notation: Prefix
Parameters: Observable Data
Returns: String
Summary: This function will return the type of the utterance of the record (i.e. spontaneous, elicitation, or repetition).

Predicates

Function Name: branching
Notation: Prefix
Parameters: Onset
Returns: Boolean
Summary: This function will return TRUE if the Onset contains more than one element, and FALSE otherwise.

Function Name: consonant
Notation: Prefix
Parameters: Feature Set
Returns: Boolean
Summary: This function will return TRUE if the Feature Set contains the attribute [consonant] and FALSE otherwise.

Function Name: diphthong
Notation: Prefix
Parameters: Syllable Nucleus
Returns: Boolean
Summary: This function will return TRUE if the syllable nucleus contains a diphthong and FALSE otherwise.

Function Name: exists
Notation: Prefix
Parameters: Any Structure
Returns: Boolean
Summary: This function will return TRUE if the structure exists and FALSE otherwise.

Function Name: fallingDiphthong
Notation: Prefix
Parameters: Nucleus
Returns: Boolean
Summary: This function will return TRUE if the structure contains a diphthong with falling sonority between the Feature Sets, and FALSE otherwise.

Function Name: hasVowel
Notation: Prefix
Parameters: Any Syllable Structure
Returns: Boolean
Summary: This function will return TRUE if the structure contains a vowel and FALSE otherwise.

Function Name: hiatus
Notation: Prefix
Parameters: Syllable, Syllable
Returns: Boolean
Summary: This function will return TRUE if there is a hiatus (heterosyllabic V.V sequence) between the two given Syllables and FALSE otherwise.

Function Name: longV
Notation: Prefix
Parameters: Syllable Nucleus
Returns: Boolean
Summary: This function will return TRUE if the syllable nucleus contains a long vowel and FALSE otherwise.

Function Name: monophthong
Notation: Prefix
Parameters: Nucleus
Returns: Boolean
Summary: This function will return TRUE if the structure contains a monophthong, and FALSE otherwise.

Function Name: risingDiphthong
Notation: Prefix
Parameters: Nucleus
Returns: Boolean
Summary: This function will return TRUE if the structure contains a diphthong with rising sonority between the Feature Sets, and FALSE otherwise.

Function Name: shortV
Notation: Prefix
Parameters: Syllable Nucleus
Returns: Boolean
Summary: This function will return TRUE if the syllable nucleus contains a short vowel and no diphthong and FALSE otherwise.

Function Name: single
Notation: Prefix
Parameters: Onset
Returns: Boolean
Summary: This function will return TRUE if the Onset has only one element and FALSE otherwise.

Function Name: stressed
Notation: Prefix
Parameters: Syllable
Returns: Boolean
Summary: This function will return TRUE if the Syllable has stress and FALSE otherwise.

Function Name: vowel
Notation: Prefix
Parameters: Feature Set
Returns: Boolean
Summary: This function will return TRUE if the Feature Set contains the attribute [vowel] and FALSE otherwise.

Relational Functions

Function Name: lessThan
Notation: Prefix
Parameters: Numeric Value, Numeric Value
Returns: Boolean
Summary: This function tests two given numeric values (including dates). If the first parameter has a smaller numeric value than the second parameter the function returns TRUE and FALSE otherwise.

Function Name: greaterThan
Notation: Prefix
Parameters: Numeric Value, Numeric Value
Returns: Boolean
Summary: This function tests two given numeric values (including dates). If the first parameter has a greater numeric value than the second parameter the function returns TRUE and FALSE otherwise.

Function Name: equals
Notation: Prefix
Parameters: Any Structure, Any Structure
Returns: Boolean
Summary: This function will return TRUE if both of the given structures are the same type and hold the exact same information. For example, if two Feature Sets have the same features or two Syllables have the same sequence of Feature Sets.

Boolean Functions

Function Name: AND
Notation: Infix
Parameters: Boolean, Boolean
Returns: Boolean
Summary: This function will return TRUE if and only if the given parameters are both TRUE and FALSE otherwise.

Function Name: NOT
Notation: Prefix
Parameters: Boolean
Returns: Boolean
Summary: This function will return the inverse of the given parameter. That is if the parameter is TRUE, FALSE will be returned and vice versa.

Function Name: OR
Notation: Infix
Parameters: Boolean, Boolean
Returns: Boolean
Summary: This function will return TRUE iff one of the given parameters is TRUE and FALSE otherwise.

4.3.4 Feature Set Searching

Many possible searches needed by researchers require content specific comparisons not possible through previously defined predicates. These searches involve specific sequences of sounds found in the IPA transcriptions. For this purpose, we will define a new predicate *contains(x, y)* which will perform the relationship $x \supseteq y$.

> **Function Name:** contains
> **Notation:** Prefix
> **Parameters:** [Feature Set | Feature Set Sequence], Feature Set
> **Returns:** Boolean
> **Summary:** This predicate will evaluate the $x \supseteq y$ (superset) relationship, where x is the first parameter and y is the second parameter of the function. It will return TRUE if x is a superset of y or x contains a Feature Set which is a superset of y, and FALSE otherwise.

Frequently, researchers may need to search for sub-sequences of features within a given Feature Set Sequence. This is done using another predicate *subsequence(x,y)*.

> **Function Name:** sequence
> **Notation:** Prefix
> **Parameters:** Feature Set Sequence, Feature Set Sequence
> **Returns:** Boolean
> **Summary:** This predicate will test to see if the sequence of Feature Sets in y can be found in x. Each Feature Set in y will have to pass the test $x \supseteq y$. Wild cards may also be used within the sequence y.

4.3.5 Feature Sets

A *Feature Set* is a mapping of an IPA glyph to a set of features representing the sound represented by the character. A list of IPA characters and their respective features can be found in Appendix B.

Specifying Feature Sets can be done either using IPA characters or explicitly specifying them. When using IPA characters to denote a Feature Set, the IPA character must be placed in double quotation marks, i.e. "a". Explicit feature sets are defined within braces and are combinations of features from the universal set U (defined below) and are separated by commas.

U = {
Vowel,Consonant,Glide,Continuant,
Voiced,Labial,Coronal,Distributed,
Anterior,Retroflex,Palatal,Velar,
Uvular,Pharyngeal,Laryngeal,Implosive,
Click,Sibilant,Nasal,Lateral,Rhotic,
Sonorant,Approximant,High,Low,
Front,Central,Back,Round,
ATR, Stop, Fricative
}

For example the feature set defined for IPA character "ʃ" using the table in Appendix X is {Consonant, Continuant, Coronal, Distributed, Sibilant, Fricative}.

Feature Sets can also be created by performing set unions. The union operator is donated by the symbol "+". This can be particularly useful when an IPA character has all the required features except a few, like in the cases of vowels with the Nasal feature added. An example of searching for the vowel "a" with Nasal feature is "a" + {Nasal}. When more that one IPA character is placed into a sequence and that sequence is OR'ed with another feature, each Feature Set in the sequence gets the added feature. For example, "ab" + {Nasal} would add the feature "Nasal" to both "a" and "b".

To complement the set union operation, another operation for removing features from a set is defined. The symbol for this operation is "-" and is used in the same manner as set union. For example, if we are searching for a set of features in the vowel "i" but do not want the feature "ATR" we would use: "i" - {ATR}.

Feature Sets may also be placed into sequences. Feature Set sequences can be expressed in two ways: a string of IPA characters inside double quotation marks; and a comma delimited list of Feature Sets between brackets. Examples respectively are "bæ" and [{Nasal},{Retroflex}].

There are situations in which mixing the notations for Feature Sets may be desirable. An example is when searching for the sound "i" followed by any sound containing the feature "Sonorant"; the corresponding Feature Set Sequence for this would be "["i", {Sonorant}]". Mixing the notation can also be useful in cases when searching for a sequence of three or more sounds and the middle sound is unimportant. In such a case it is important to remember that the empty set "{}" is a subset of every set. Searching for the sound "a" followed by an uninteresting sound and then the sound "b" would yield the Feature Set Sequence "["a", {}, "b"]".

4.3.6 Using Contains and Sequence

There are two possible ways to use *contains(x,y)*:

- *x* is a single Feature Set, *y* is a single Feature Set
- *x* is a Feature Set Sequence, *y* is a single Feature Set

where *x* is the first parameter and *y* is the second parameter to the function.

In the case where both parameters are single Feature Sets, the predicate simply returns whether *x* is a superset of *y*.

Example	***Result***
contains({Nasal,Consonant}, {Nasal})	TRUE, since {Nasal} is a subset of {Nasal,Consonant}
contains({Nasal,Consonant}, {Nasal,Consonant,Lateral})	FALSE, since the second parameter has the extra feature [lateral]

In the case where the first parameter is a Feature Set Sequence and the second parameter is a single Feature Set, the entire Feature Set Sequence is checked as to whether it contains the given Feature Set.

Example	***Result***
contains([{Nasal,Consonant}, {Consonant, Lateral}], {Consonant})	TRUE, since {Consonant} is a subset of both {Nasal, Consonant} and {Consonant,Glide}
contains([{Nasal,Consonant}, {Consonant,Lateral}], {ATR})	FALSE, since {ATR} is not found in either {Nasal, Consonant}, or {Consonant, Lateral}

The sequence predicate is used to test for sequences of Feature Sets within another given Feature Set. Since the user may not always know the entire given sequence of features, wild cards may be used to place separation between the desired Feature Sets. There are two wild cards that may be used inside the given Feature Set Sequence:

- $ - used to match one Feature Set; and
- * - used to match zero or more Feature Sets.

For example, if we wanted to search for a sequence of three features, but did not care what the middle feature was, we would use the "$" symbol in place of the second feature. As well, if we wanted to see if two features existed in a sequence, but didn't care if there were any features separating them, we would use the "*" between the two relevant features. Usage of the sequence function is illustrated below.

Example	*Result*
sequence(“abc”, “ab”)	TRUE, since “ab” is a substring in “abc”
sequence(“abc”, “ac”)	FALSE, since “ac” is not a substring in “abc”
sequence(“abc”, “a$c”)	TRUE, since the wild card will match the “b”
sequence(“abc”, [“a”, {}, “c”])	TRUE, since “a” and “c” represent the same set in both parameters, and {} (the empty set) is a subset of every set.
sequence(“abc”, “a*b”)	TRUE, since the wild card “*” will match zero or more features between “a” and “b”
sequence(“abcd”, “a*d”)	TRUE, since the wild card “*” will match the “bc” in the given sequence

4.3.7 Variable Quantification: *foreach*

During query evaluation, it will be desirable to select a set of records from the corpus, or the set of syllables in the target and child phrases of a record. These tasks are accomplished using the *foreach* loop syntax. The basic format of *foreach* is:

```
foreach <variable>[,<variable>]*: <type>[,<type>] in <location>[,<location>]*
    <QUERY>
end for
```

where *variable* is the identifier requested, *type* is the data type for the variable and *location* is the parent structure of the variable.

Since queries are designed to return a set of Records matching given criteria, almost every query will start with a *foreach* block indicating selection of records from a particular corpus. This is done using the following statement:

```
foreach r: Record in corpus(“Main”)
```

This allows for each Record in the "Main" corpus to be accessed using the defined variable *r*.

Now that we have a means of selecting components of records in a given corpus we can then select the syllables from the target and actual phrases. The statement

```
foreach t,a: Syllable,Syllable in targetPhrase(r), actualPhrase(r)
```

will assign the aligned syllables for the target and actual phrases to *t* and *a* respectively. This is the most convenient method for accessing the syllables, however it is also possible to compare every target syllable to each actual syllable and vice versa using two nested *foreach* blocks as shown below.

```
foreach t: Syllable in targetPhrase(r)
    foreach a: Syllable in actualPhrase(r)
```

4.3.8 Custom Predicates

When a particular query becomes useful more than once, the query can become a *custom predicate.* Custom predicates allow long and/or useful queries to be used as predicates in future queries. Several of the predefined predicates used in the query language are created in this way. The general syntax for defining a custom predicate is

```
let <PredicateName>(<ParameterList>):= <Query>
```

An example of a system predicate defined in this way is *consonant(x)*, which is defined by the system as:

```
let consonant(x) := contains(x, {Consonant}).
```

This predicate simply states that a Feature Set is a consonant if and only if it contains the feature “Consonant”. While the above example is simple, any valid query can be turned into a predicate, thus making the query language infinitely expandable.

4.3.9 Query Grammar

Grammars are a means of defining the syntax for a language. A grammar has a *start symbol* which is the point syntax validation must begin. The following variant of an Extended Backus-Naur Form grammar displays the syntax of the query language detailed in this chapter. The start symbol is `Query`.

```
Query           ::= (DefinitionList ';')(Quantifiers ';')
                        O-query ('OR' O-query)*
DefinitionList  ::= Definition(',' Definition)*
Definition      ::= 'let' Identifier '(' ParamList ')' ':=' Query
Quantifiers     ::= 'foreach' Identifier (',' Identifier)* ':' Type 'in'
                        Function (',' Function)*
                            (',' Quantifiers )*
O-Query         ::= A-query ('AND' A-query)*
A-query         ::= ('NOT')? Function
Function        ::= Identifier '(' ((Function)*|ParamList) ')'
ParamList       ::= Identifier (',' Identifier)*
IPASet          ::= Char (SeqOp Char)*
FeatureSequence ::= '[' IPASet (',' IPASet)* ']'
Char            ::= '"' ( ipachar )* '"' | FeatureSet
FeatureSet      ::= '{' Feature (',' Feature )* '}'
SetOp           ::= '+' | '-'
Identifier      ::= [a-zA-Z][a-zA-Z0-9_]*
Integer         ::= [1-9][0-9]*
Feature         ::= 'Vowel' | 'Consonant' | ...
Type            ::= 'Record' | 'Phrase' | 'Syllable' |
                    'Onset' | 'Nucleus' | 'Coda' |
                    'Appendix' | 'Integer' | 'Date' |
                    'String' | ...
```

Table 4.2 Query Grammar

4.3.10 Queries for Syllable Processes

This section will introduce queries for the syllable processes found in section 2.2.2. When performing experiments aimed at child language acquisition, many processes often re-occur. Many of these processes can be detected using relatively short (relative to some or the more complex queries that can be constructed) queries. Repeatedly building these queries is not always practical. For this reason, our system looks at providing a set of twenty-four of these commonly occurring processes. These twenty-four processes are extended from the original *ChildPhon* interface and encapsulate the typical queries of interest to linguists. When records are selected for viewing (i.e. while searching through the database), all of the processes presented below will be instantly detected, without requiring an explicit request from a user of the system, to provide quick summaries of some of the patterns or processes occurring.

We now present these built-in queries. The following queries assume that the variables *r, t,* and *a* represent the current Record, Target Syllable, and Actual Syllable respectively.

Structure	Syllable
Process: Syllable Truncation **Query:** exists(t) AND NOT exists(a)	

Structure	Syllable Nucleus
Process: Long Vowel Shortening **Query:** long(nucleus(t)) AND short(nucleus(a))	
Process: Short Vowel Lengthening **Query:** short(nucleus(t)) AND long(nucleus(a))	
Process: Diphthong Reduction **Query:** diphthong(nucleus(t)) AND monophthong(nucleus(a))	
Process: Short Vowel Diphthongization **Query:** short(nucleus(t)) AND diphthong(nucleus(a))	
Process: Long Vowel Diphthongization **Query:** long(nucleus(t)) AND diphthong(nucleus(a))	
Process: Vowel Epenthesis **Query:** NOT hasVowel(t) AND hasVowel(a)	
Process: Vowel Deletion **Query:** hasVowel(t) AND NOT hasVowel(a)	

Structure	Vowel-initial phrase
Process: Vowel Deletion **Query:** vowel(pos(t, 1)) AND NOT exists(a)	
Process: Consonant Epenthesis **Query:** vowel(pos(t, 1)) AND consonant(pos(a, 1))	

Structure	Hiatus
Process: Hiatus Reduction **Query:** hiatus(t, nextSyllable(targetPhrase(r),t)) AND (hasVowel(nextSyllable(targetPhrase(r), t)) AND NOT hasVowel(nextSyllable(acualPhrase(r), a))	
Process: Consonant Epenthesis **Query:** hiatus(t, nextSyllable(targetPhrase(r), t)) AND (vowel(pos(nextSyllable(targetPhrase(r), t), 1)) AND consonant(pos(nextSyllable(actualPhrase(r), a), 1)))	

Structure	Vowel-initial syllable
Process: Vowel Deletion **Query:** vowel(pos(t,1)) AND consonant(pos(a, 1)) AND similar(pos(t, 2), pos(a,1))	
Process: Consonant Epenthesis **Query:** vowel(pos(t, 1)) AND consonant(pos(a,1)) AND similar(pos(t,1), pos(a,2))	

Structure	s-Consonant(sC) Clusters
Process: s deletion **Query:** exists(lappend(t)) AND NOT exists(lappend(a))	
Process: C deletion **Query:** NOT branching(onset(t)) AND equals(lappend(t) , onset(a)) OR branching(onset(t)) AND equals(lappend(t) , pos(onset(a), 1))	
Process: Whole cluster deletion **Query:** exists(lappend(t)) AND exists(onset(t)) AND NOT exists(onset(a))	
Process: Vowel prothesis **Query:** equals(lappend(pos(t, i)) , coda(nextSyllable(actualPhrase(r), a))) AND NOT exists(previousSyllable(targetPhrase(r), t))	
Process: Vowel Epenthesis **Query:** exists(lappend(t)) AND NOT exists(lappend(a)) AND NOT exists(previousSyllable(targetPhrase(r), t)) AND similar(onset(previousSyllable(actualPhrase(r), a))), lappend(pos(t, i)))	

Structure	Branching Onsets
Process: C1 Deletion **Query:** branching(onset(t)) AND single(onset(a)) AND similar(pos(onset(t), 2) , pos(onset(a), 1))	
Process: C2 Deletion **Query:** branching(onset(t)) AND single(onset(a)) AND similar(pos(onset(t),1) , pos(onset(a), 1))	
Process: Vowel Prothesis **Query:** branching(onset(pos(t. i))) AND branching(onset(pos(a, i))) AND NOT exists(coda(previousSyllable(actualPhrase(r), a)))	
Process: Vowel Epenthesis **Query:** branching(onset(pos(t. i))) AND NOT branching(onset(pos(a, i))) AND similar(onset(nextSyllable(actualPhrase(r), a)) , coda(pos(t, i)))	

Structure	Branching Onsets
Process: Vocalization of 2nd C **Query:** branching(onset(t)) AND single(nucleus(t)) AND NOT branching(onset(a)) AND diphthong(nucleus(a))	

4.4 Reports and Report Generation

This system will generate human readable representations of data contained within a given corpus or as the result of a query. The reports may also be generated from statistical information such as the occurrence of a specific phonological process (e.g. Branching Onset Reductions) over time.

There are three report types for which this system will be designed to create:

- Text Reports

  These reports include all of the observable data, transcription data, and automatic parses found within a given corpus or a subset of a corpus – such as one created from a query. The data will be presented in UTF-8 text encoding and formatted in a tabular organization.

- Bar Charts

  These reports provide a graphical representation of the prominence or a linguistic process in the form of varying length bars in columns. An example of a bar chart, created from the occurrences of Syllable Truncation over a span of two years is shown below.

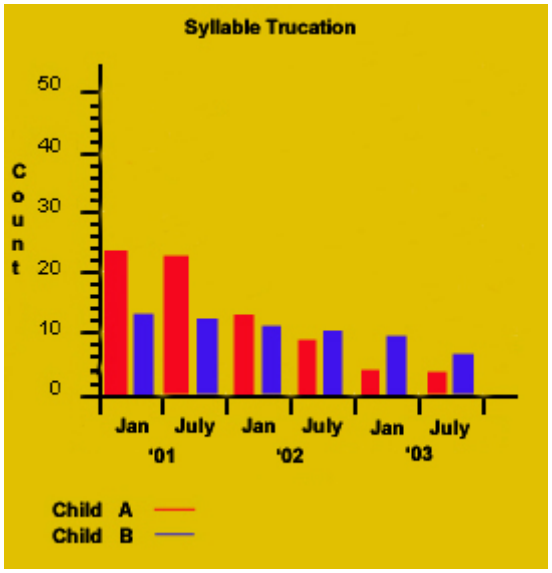


Figure 4.2 Bar Chart Report

- Line Charts

  These reports provide a graphical representation of the longitudinal display of a given process over a specified period of time. This type of report can also display the relationship between two different processes over time. An example of a line chart, created from the occurrences of Onset Reduction over a span of seven months for two given children.

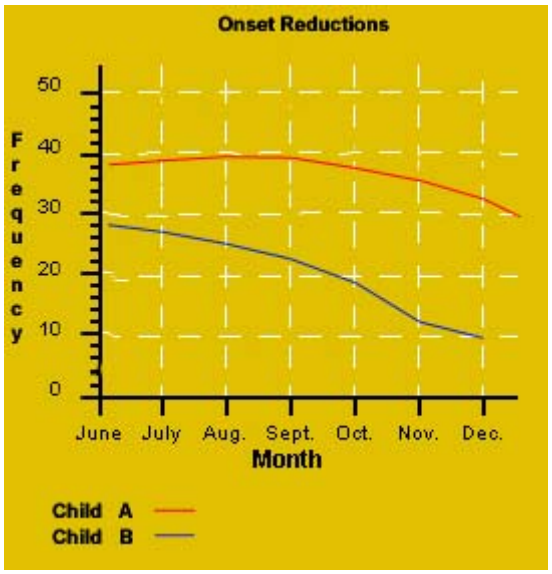


Figure 4.3 Line Chart Report

Chapter 5: XML Document Definition

This chapter discusses the structure of the XML document used for the accompanying software package. The formal definition of the schema is also presented.

5.1 Overview

We seek not to implement our own XML scheme, but extend a currently existing scheme, in the hopes that other applications will do the same. This would allow easier integration and extensibility of application modules.

We will be building upon the schema designed by the *Talkbank* organization as discussed earlier in this document. While the complete Talkbank schema is quite extensive, we will only present the elements and attributes necessary to incorporate our additions.

The motivation for the proposed additions comes from the way information is organized and processed in our system. We mentioned a tier-structure in chapter 2. The following section will define this structure and illustrate how it develops during a linguist's research.

Following this discussion, we will detail the proposed additions to the *Talkbank/CHAT* XML schema.

5.2 Tier-Oriented Data Structure

Phonological analyses, as mentioned earlier, are carried out on systematic comparisons of syllables between their target and produced forms. We use linguistic software to process raw data, in the form of audio/video recordings, and produce a data representation that can be easily analyzed. However, the processed information must be stored in a way that does not destroy any of the previous data. Simply put, we want to take an autonomous layer of information and build upon it, creating another layer of information that is more refined or relevant for study. We call these abstract autonomous layer *tiers*.

By providing a tier-oriented structure, we can process one tier to create a new tier (or add to an existing tier) without changing any of the data up to that point. This process is desirable as researchers will often need to go back and study earlier versions of record. Also, errors may arise and the problem may originate from a "lower" tier.[15] Invalidating this tier removes all subsequent data.

The tier structure developed here stems directly from the use cases presented in Chapter 3. Tracing the first seven use cases we can isolate six identifiable tiers. These tiers are illustrated in Figure 5.1 below. We will be referring back to this figure for the remainder of this section.

[15] In this section we will regard tiers that are composed before another tier as being *below* it, or *lower* than it in the tier structure. The figures illustrate this convention.

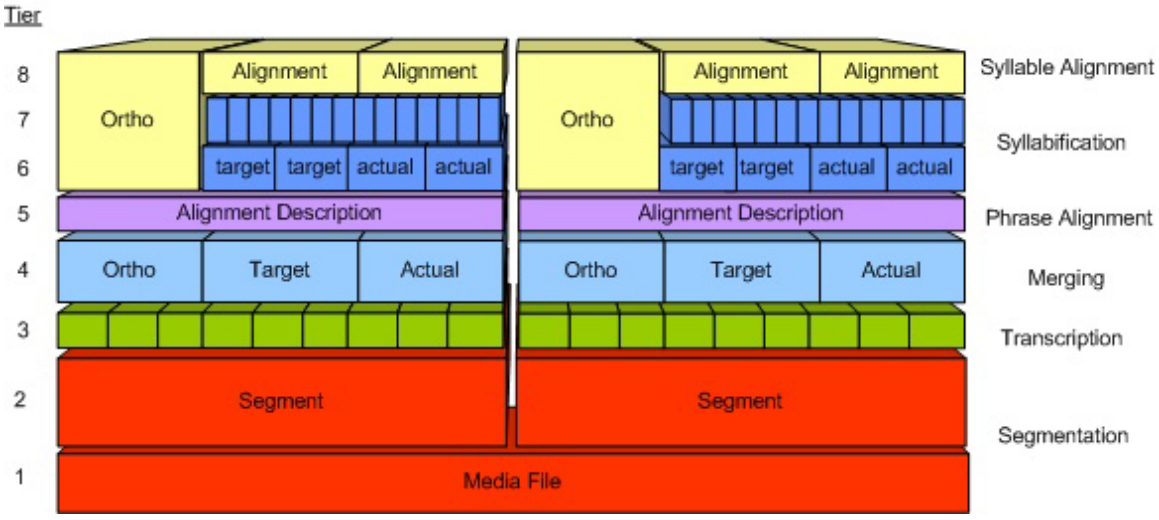


Figure 5.1 Linguistic Data-Tier Organization

Media File – Tier 1

The multimedia files that provide the raw observational data for the experiments. Media files are the main source of information for many linguists. From this foundation we be can develop the other tiers.

Segment – Tier 2

From the "SegmentSpeech" use case we identify the result of segmentation as a *segment*. The figure above shows only two segments from a single file; in practice there could be dozens.

Transcript – Tier 3

From segments we get *transcripts* which result from the "TranscribeSegment" and "TranscribeAdultChildSegment" use cases. Since multiple transcribers can transcribe a single segment as they see it, we must store multiple transcriptions.

Merged Transcripts – Tier 4

Merging is done on these transcriptions resulting in a single orthographic, target, and actual utterance *transcription tuple*. This stage is a good illustration of how previous tiers are retained for future study (or recovery, in the event that an error is noticed later).

Alignment Data – Tier 5

The "AlignPhrases" use case indicates that an aligner uses a transcription tuple to create an *alignment description*. This tier describes how to retrieve phrases from the tier below it in pair-wise fashion.

Phrase Alignment and Syllabification – Tiers 6 & 7

Phrase alignment breaks utterances into phrases. The phrases come from the alignment description. Syllabification is performed automatically on the *target and actual phrases* to generate *syllables* (the small blue blocks above phrase alignment).

Syllable Alignment – Tier 8

Automatic alignment is then performed on the set of syllables from tier seven to create a *syllable alignment*. The aligned syllables are then visible for analysis.

These tiers, paired with the use cases that use/create them, reflect the information migration from raw media files to analyzable syllables. An additional tier could be added to Figure 5.1 that described automatic syllable parses and the more common processing done on aligned syllables. This information is not stored in the database (our XML files). It is generated "on-the-fly" when records are requested by users of the system. It is not, therefore, included here.

Tiers are representative of the nesting structure observed in XML files. It is for this reason (among others presented in Chapters 2 and 3) that XML will be used to store this information. The following section describes the additions proposed to *Talkbank's* XML scheme for organizing linguistic data.

5.3 Proposed Schema Additions

To encourage compatibility with existing and future linguistic software systems, the XML storage for our software utilizes an existing XML schema. This means that data collected by *CLAN* can be easily shared with our software, and vice versa. *Talkbank's* scheme currently does not provide facilities for storing phonological data, besides IPA transcriptions. However, it does have a means of storing users, participants and metadata within the system. These facilities will be used by the software described in this thesis.

5.3.1 Reused Schema Elements

The first-level elements from the *CHAT* structure that will be used are given below in schema format.[16]

The notation and colour conventions of this schema representation:

- a choice: [x y z] ; a sequence: (x y z)
- Colours used: **{type declarations}**; element names attribute names ; restriction/extension facets; **{links to type definitions}**; comments;
- an extension of a type is {<<type>>} ; and a restriction to it : {>>type<<}

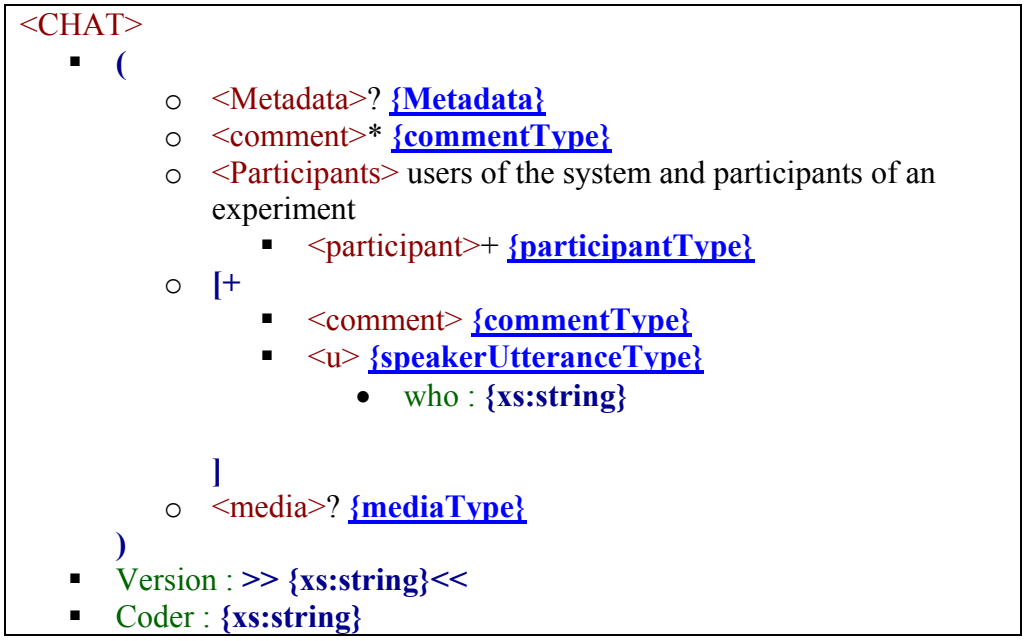


Figure 5.2 CHAT XML Schema

The <Metadata> elements are used exactly as one would expect. They provide a way of adding session information to the database about a particular experiment and its participants. The <Participants> elements list both the users of the system, as well as the experiment participants. In this regard, <Participants> is interpreted to mean the participants of the system, not necessarily the participants of the experiment.

A database can have any number of utterance records. The <u> elements denote these utterances. The "who : **{xs:string}**" attribute indicates the participant that made the

[16] Note that this is not the complete scheme. For a comprehensive discussion of Talkbank's scheme, see http://xml.talkbank.org:8888/talkbank/talkbank.html.

utterance. This attribute list will be augmented by a *multiplexing attribute* for denoting record type.

5.3.2 Schema Additions

This multiplexing is achieved by adding a single multi-value attribute to the <u> element:

"type : **{xs:string}**".

The type of a record does not indicate the utterance type, by the record type. For example, to delimit a record as one for phonological analysis, we could use:

<u who="name" **type= "phon"**>

…

</u>

This creates an environment for multiple record types to exist within the same database. For existing software systems, like *CLAN*, this is where the only difference would lie. Parsers using such an XML file would first look at the "type" field to determine if it is relevant to its analysis.

With this type field, we can now define a structure that identifies a phonological record. The elements, their nesting and attributes reflect the tier structure we presented above. It should be noted that this addition to the scheme will not affect *CLAN*'s processing of these files since all the phonological record-specific information is contained within the <u> element. Once a parser sees this record is not the appropriate type, it can bypass it and continue.

The following details this organization in XML schema format. We include only those elements we look to add to the new scheme.

<CHAT>
- (
  - **[+**
    - <u> **{speakerUtteranceType}**

  ]

)

Global types

- **speakerUtteranceType :**
  - **<< {utteranceType} >>**
    - type :
      - **>> {xs:string}<<**
        morpho to be used in CLAN
        phon to be used in our software

- **utteranceType :**
  [? one of the following blocks, based on *type* of utterance
  - **(**
    - (*Talkbank's* existing utteranceType definition)

)
  - **(**
    - <ortho> orthographic transcription
      - type : **{xs:string}**
    - <phrase>+
      - <transcript>+ **{transcriptType}**
      - <syllabification> **{syllabificationType}**
      - type :
        - **>> {xs:string}<<**
          target
          actual
    - <alignment>? **{alignmentType}**

)

  ]

- **transcriptType :**
  - <ipa> full transcribed phrase
  - who : **{xs:string}** the transcribers participant id

- **syllabificationType :**
  - <syllable>+ **{syllableType}**
  - who : **{xs:string}** points to the transcription this syllabification was taken from

- **syllableType :**
  - <ipa>
    - <constituent>

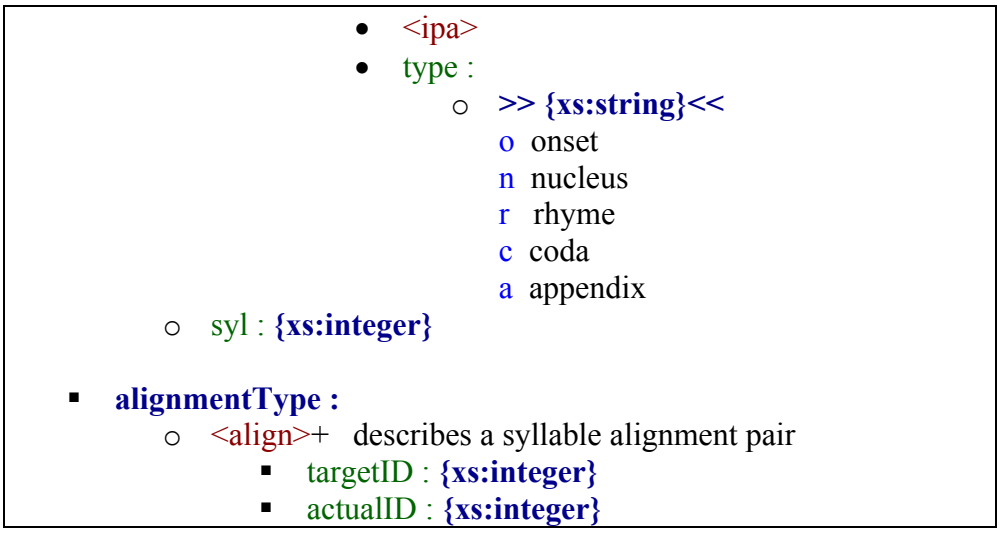


Figure 5.3 Proposed CHAT Schema Additions

This structure is used to define an XML file that can be parsed by the system to generate a set of records of the appropriate type (in this case, phonological records). All of the required data, as specified in the use cases and system requirements in Chapter 3, has been accommodated in this scheme. If we look back at the list of searchable record information from section 4.2.1 we see that Observational Data is covered in the <Metadata> tags, or can be derived from it. The Phrase information is coved by the <phrase> element, as is the Syllable information.

The *Talkbank* scheme, together with the proposed "type" attribute, provides a foundation for defining additional linguistic records. Adding new record types is as simple as defining the record type identifier – type="morpho" for morphological records, or type="morphSyn" for morpho-syntactic records – and defining the structure that these records must take.

Appendix A gives a sample XML file for storing both phonological and morphological records (Talkbank, 2001). This file contains three records from a sample media file.

With a scheme defined, we can now continue to characterize the architecture of a system based on this scheme.

Chapter 6: System Architecture

6.1 Subsystem Identification

The architecture of this system can be broken into eight (8) high-level components detailed in the following sections. A top level view of the system and the module interactions can be viewed in Figure 6.1.

This division into modules fosters reused functionality across modules while allowing minimal dependency upon the development of other modules. For example, the query language can be used by any module wishing to use it, but still remains separate from everything else in how it is implemented. Associations and interfaces between modules are abstracted, allowing inter-module design to remain largely decoupled [13].

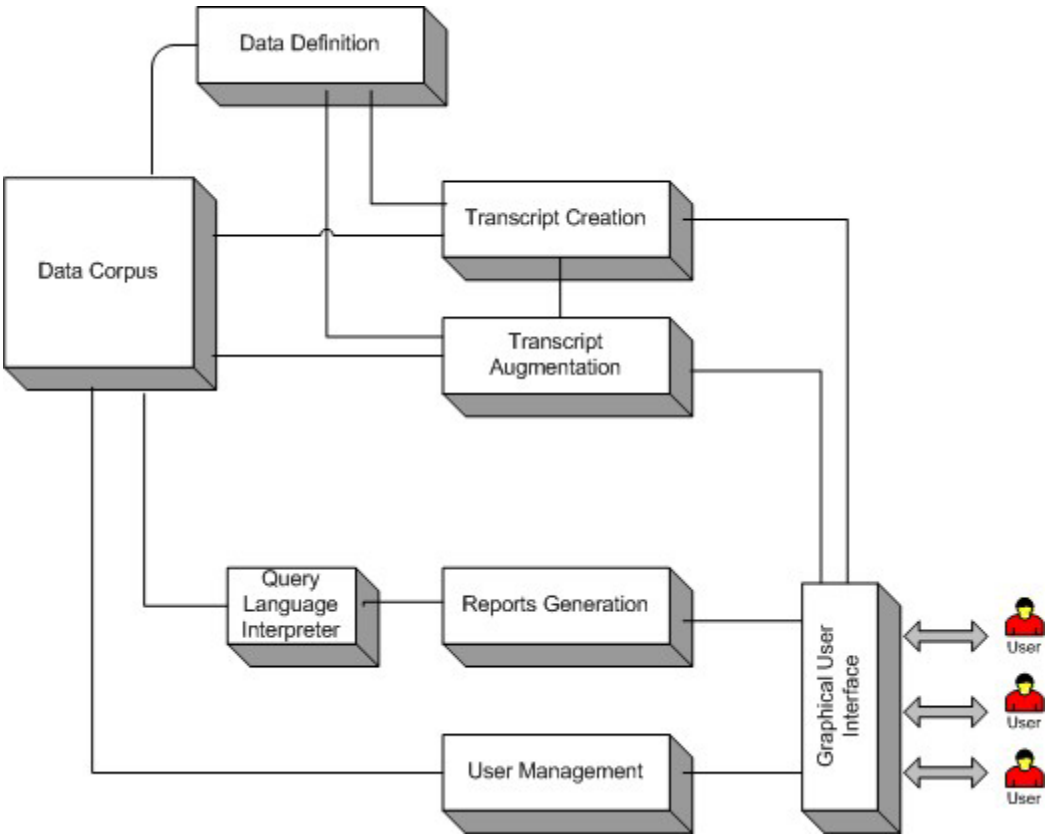


Figure 6.1 Architectural Design Model

6.1.1 Data Corpus

Responsibilities and Goals

The data corpus provides a means of organizing system data such that the query interpreter, discussed shortly, can effectively and efficiently retrieve data from it. This organization is the basis of the entire system whereby all major components interface with the data corpus to add or retrieve data, or analyze its structure.

This organization provides an aid for the User Management package described below in Section 6.1.7. By storing information on users known to the system and the records they interact with, controlled (and ultimately, restricted) access is made easier. This builds a layer of concurrency control and privileged viewing rights between these two components.

The database corpus controls input and output between data stored on disk and in-memory data which is manipulated by the application directly. The disk's data is also used in exporting data to other systems.

Persistence is another responsibility of this module and is achieved by providing simple locking mechanisms on the central data repository. The primary goal of this module is to make storage and data movement throughout the system easy and consistent.

As this is a supporting module and does not directly interface with the user, it does not directly correspond to any use case defined in the system requirements. It is included in the software design model because of its significance. Explanation of other subsystems would be meaningless without the database system.

Class Diagram

The following shows the object diagram for the data corpus subsystem. It is noteworthy to mention that the RecordIterator is simply derived from java.util.ListIterator. This is used to encapsulate record access.

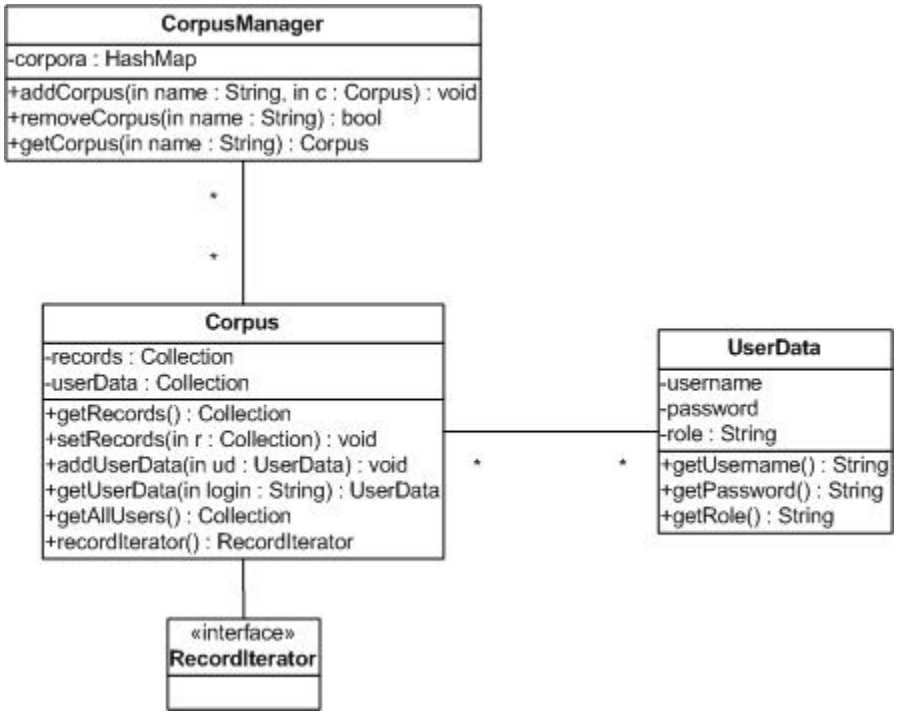


Figure 6.2 Data Corpus class diagram.

The system can import and export Corpus objects. This is a feature of the data corpus module. Input/Output is a package contained within the data corpus module. This organization can be viewed as in Figure 6.3.

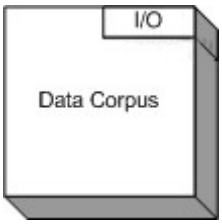


Figure 6.3 I/O package contained within the Data Corpus subsystem.

The class diagram for this package follows.

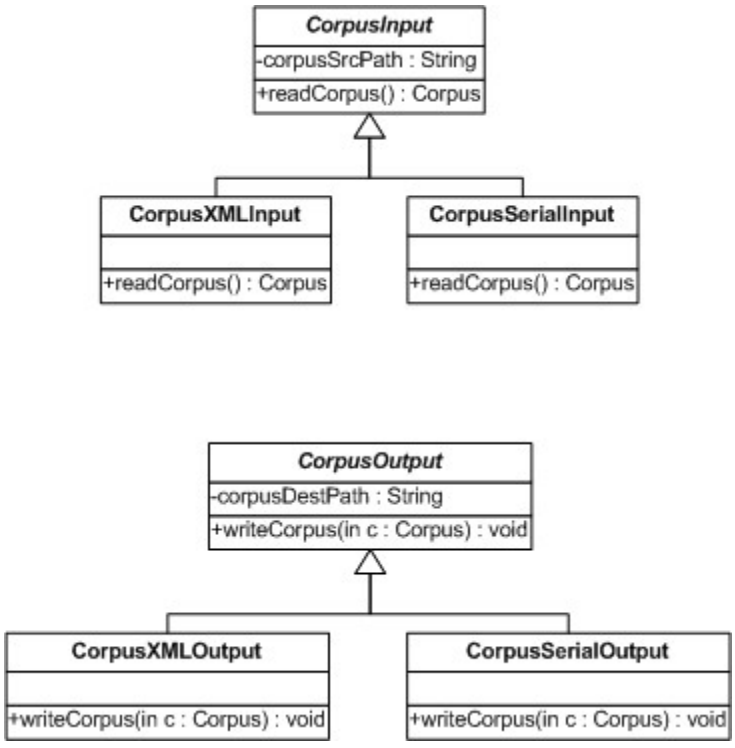


Figure 6.4 "Import/Export subsystem" within the Data Corpus subsystem.

6.1.2 Data Definition

Responsibilities and Goals

While the database corpus in the previous section was the organization of the system data, the data definition is the embodiment of that data. This subsystem deals only with representing the objects residing in the system. Attributes of the fundamental components of transcripts, states of attributes, and actions on those attributes are the focus.

This subsystem is relatively stand-alone in that in invokes no tasks within the system on its own. It is simply a way of maintaining the steady state of the system. Functions on data objects are invoked by external system actors.

Like the database corpus, there are no direct origins of this subsystem in the elicited use cases. This subsystem is also a support system. In contrast to this, the subsystems of Sections 6.1.3 to 6.1.8 are heavily grounded in their respective use case derivations.

Class Diagram

Objects within an object-oriented system can be categorized as one of: entity – persistent information tracked by the system, control – represents tasks that performed by actors and are performed on entity objects, and boundary objects – visualizations of interactions between an actor and the system [4].

The data definition is a collection of the persistent objects that are built and operated on within the system. These are all entity objects. The organization of these objects can be seen in Figure 6.5 below.

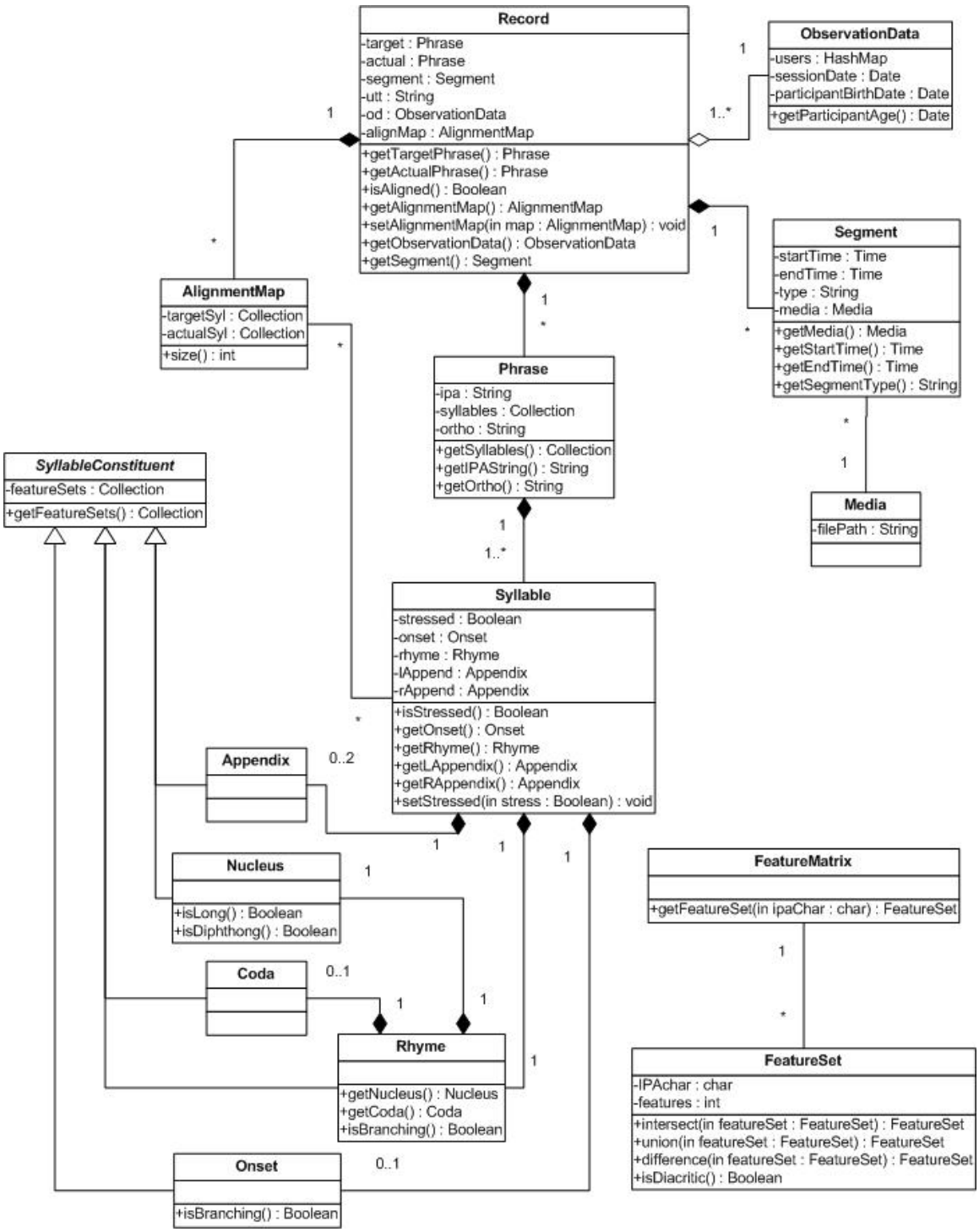


Figure 6.5 Data Definition module class diagram

6.1.3 Transcript Creation

Responsibilities and Goals

Creation entails the elaboration of a transcript to incorporate the information that defines it. The main goal of this module is to cumulatively gather all the available data for a record into that record.

Transcription creation is responsible for extracting designated segments of interest data, as identified by a user, from media files. From these segments, transcripts are made. Transcripts are further developed by orthographically transcribing the verbal information found in the media segments. This responsibility, as well as transcribing orthographic data into its IPA equivalent, falls on the transcription creation subsystem.

Creation further builds transcript records by collecting external metadata from the user as it pertains to the user and/or participants of a study, the media information involved, and the session data.

Culmination of the creation responsibilities is identified by the need to notify the data corpus of the fully or partially created transcript information for storage.

The use cases that drive the modulation of this subsystem are:

- SegmentSpeech
- TranscribeSegment
- TranscribeAdultChildSegment
- IPAFromTargetOrthography

These use cases deal with users creating and building upon transcripts. Subsequent augmentation is then performed on the data to prepare it for analysis. The responsibility suite that deals with editing transcripts without really adding anything new is covered next.

Class Diagram

To allow easy addition of future media types and players, the multimedia package of the system is designed using the *AbstractFactory* pattern. This lets the *MediaController* request a media player and let the 'Factory' determine what player to use. This is illustrated below.

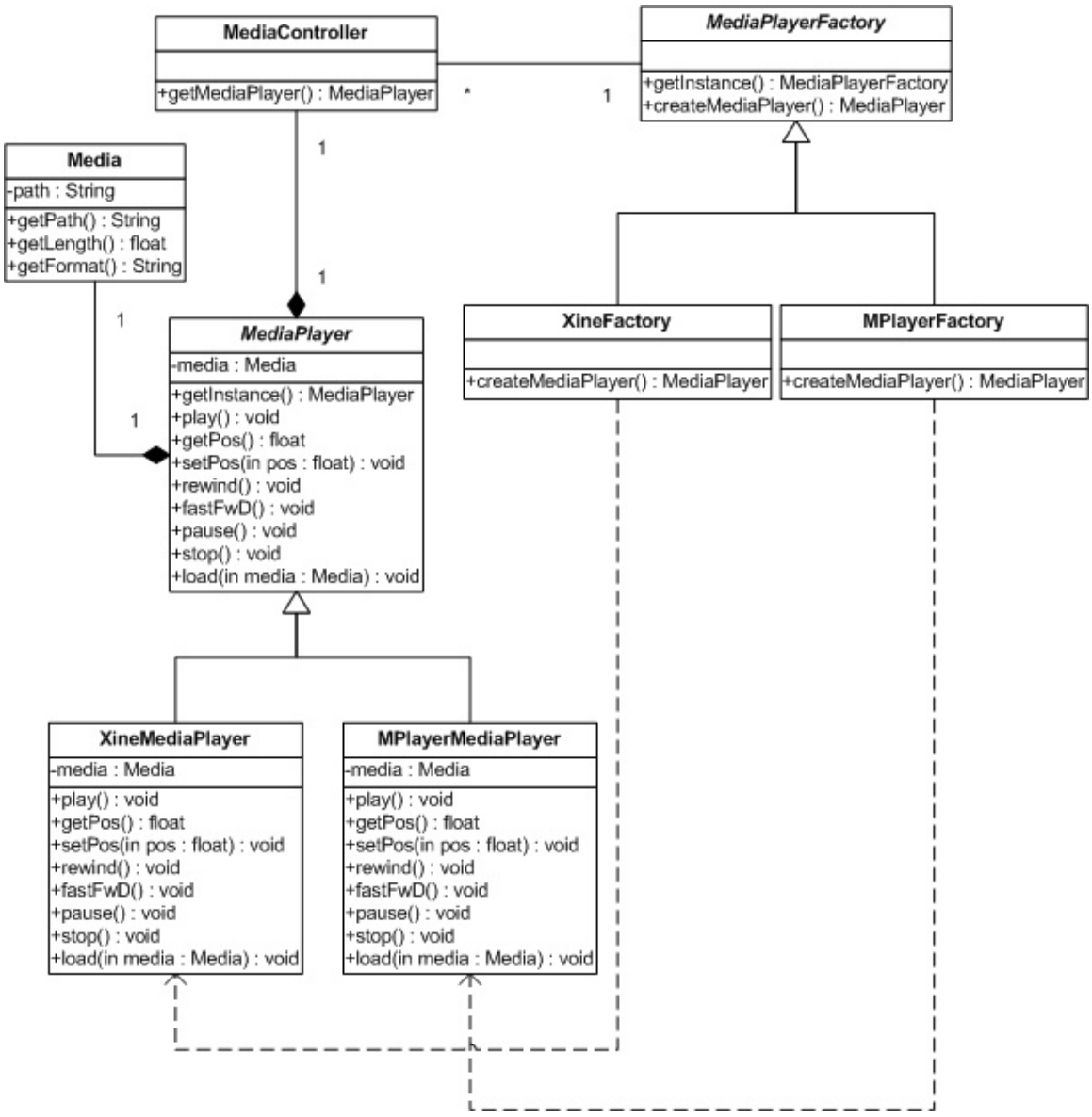


Figure 6.6 Media support organization.

Here, the *MediaController* creates a new Factory that will manufacture *MediaPlayer*s. The *MediaController* is oblivious to the kind of concrete factory used or the *MediaPlayer* instance produced. This is determined by either user preferences or system properties, or a combination of both.

The media player package is used by the software for segmentation and later for transcription. The following figure shows the segmentation module that will be provided by the system.

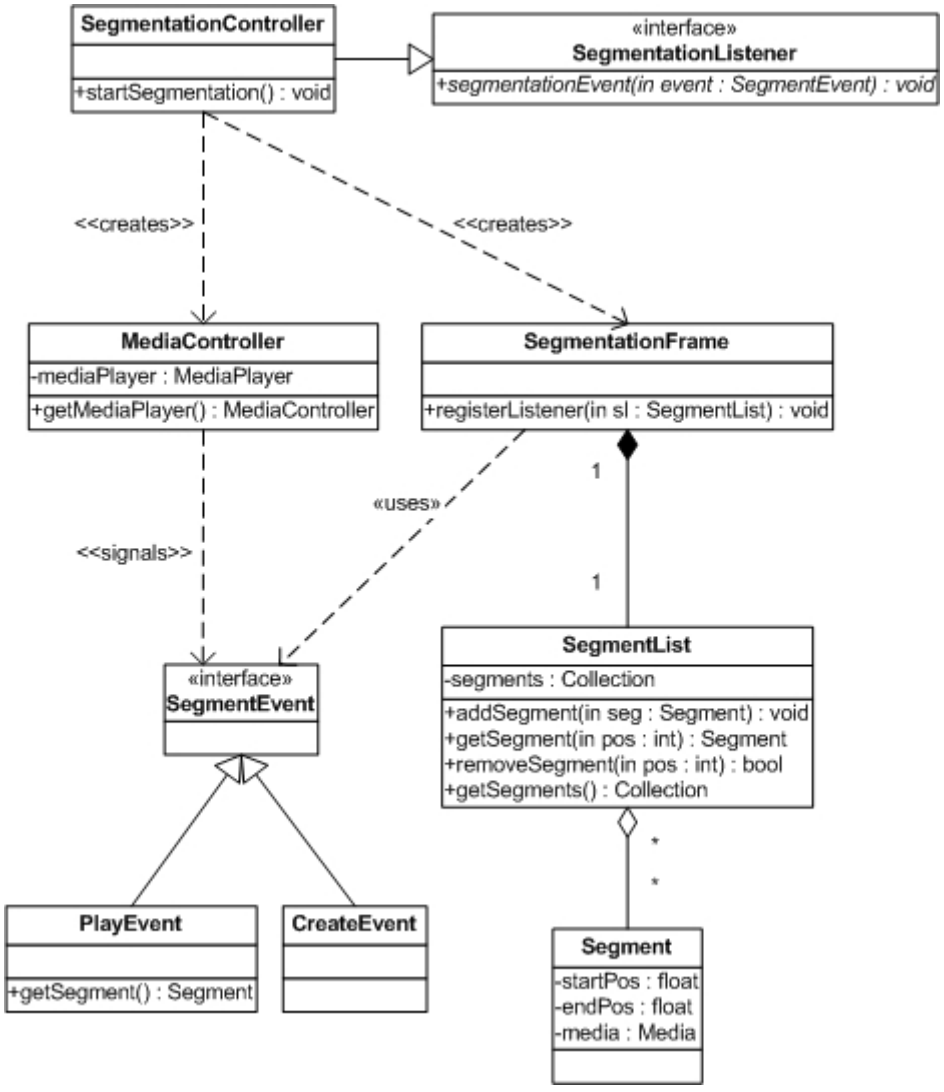


Figure 6.7 Segmentation Module

6.1.4 Transcript Augmentation

Responsibilities and Goals

Augmentation provides a means of further developing raw transcripts into evolved records of analyzable data.

Its first responsibility is to facilitate merging of similar transcripts into a single transcript providing the highest level of accuracy. This makes alignment across phrases possible. Alignment is also dealt with in the transcript transformation subsystem. Merged records are divided into phrases by this system. Alignment between target and actual phrases within a transcript is performed within the confines of this system.

The task of aligning syllables within matching sets of phrases across target and actual is done by the augmentation system, as set out by the 'AlignSyllables' use case. This involves performing pattern matching and, in some cases, approximations. The correctness of this is then required by the 'ValidateSystemParses' use case.

A means of validating system parses – proper syllabification and built-in system analyses – is delegated to the augmentation module. Confirmation from a user notifies this module to make a record available for analysis.

The responsibilities detailed above correspond to those requirements as set out in the following use cases:

- Merge
- AlignPhrases
- AlignSyllables
- ValidateSystemParses

Class Diagram

The Transcription process was discussed earlier in this thesis. The following diagram represents the software implementation of this process.

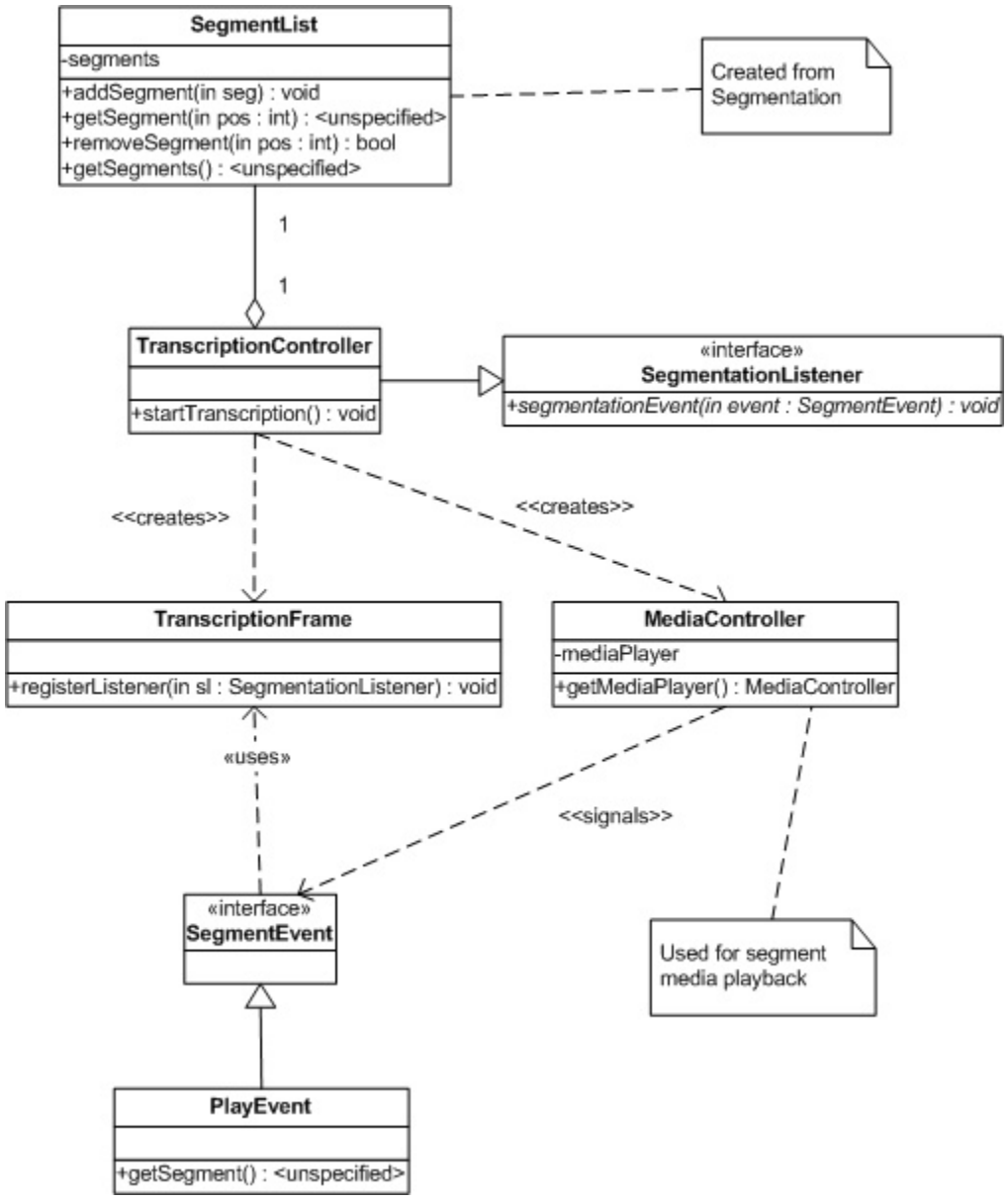


Figure 6.8 Transcription class diagram.

6.1.5 Query Language Interpreter

Responsibilities and Goals

Logistical analysis of a user query is the responsibility of the query interpreter subsystem. This unit must systematically convert a request from a user, in the form of a query, into a set of records from the database that match query-specific parameters.

As a result of this need to determine parameters, decomposition of queries into searchable criteria to use in matching records within the data corpus must also fall on this subsystem. When queries are passed to the query interpreter, intermediate parsing into a syntax the system can handle must first take place.

The query interpreter system determines what searchable contexts are valid. These were detailed in Section 4.2. How queries are compiled from these contexts is covered in Section 6.1.8: Graphical User Interface.

The need for a query interpretation system comes out of the 'SelectFromCorpus' use case in Section 3.3.1.

Class Diagram

The process of interpreting queries is broken down in to 4 parts:

- Syntax validation – the system must determine that the syntax of the given query meets the requirements of the query language grammar
- Query tree compilation – query *strings* are not interpreted. These strings are instead used to build trees structures that represent the semantic meaning to the query.
- Semantic validation – trees are built using only the rules of the grammar. Query trees can then be validated to detect errors resulting from invalid predicate or selector names, or the invalid arrangement of *nodes* within a tree.
- Query interpretation – runs query tree against each record to find matching records.

Figure 6.9 illustrates the objects involved with this process. The following diagram, Figure 6.10 illustrates the flow of events though these objects and indicates how a query is run, or how errors are detected.

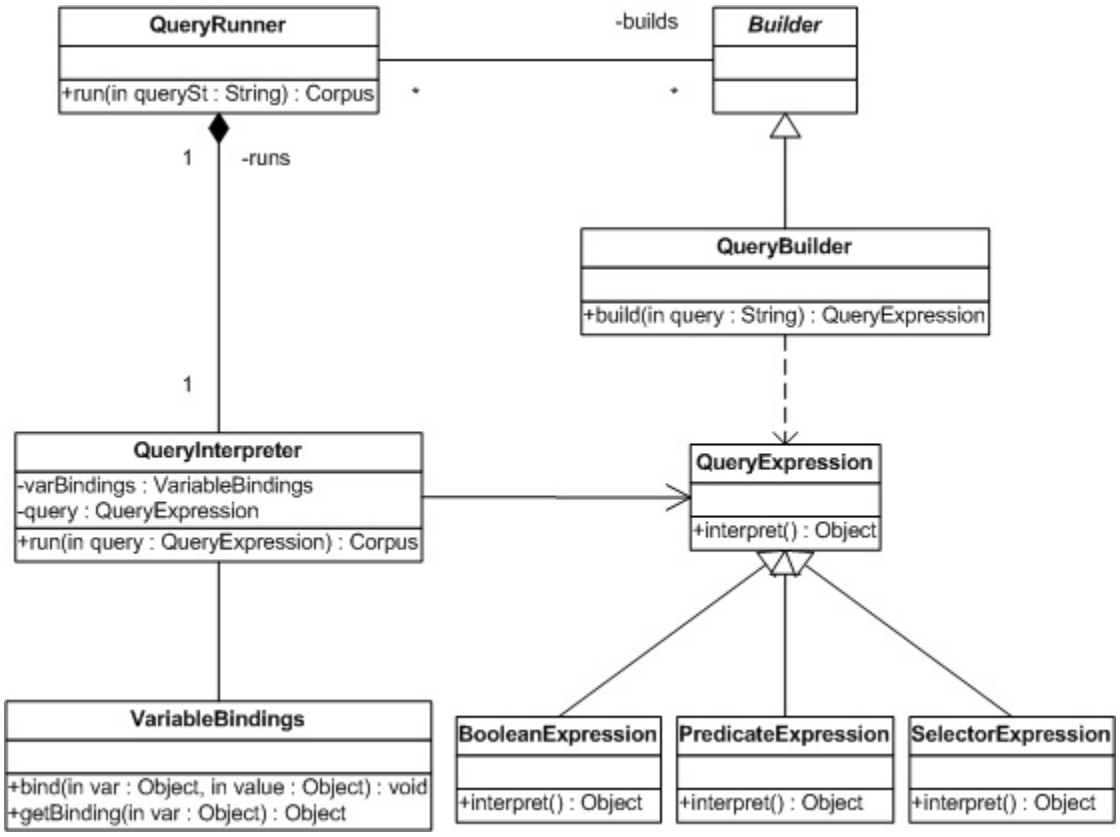


Figure 6.9 Query Language Interpreter Class Diagram

Activity Diagram

The following diagram illustrates how the steps described above take place and how the results are returned. It shows the possible flows that the query interpretation process can take.

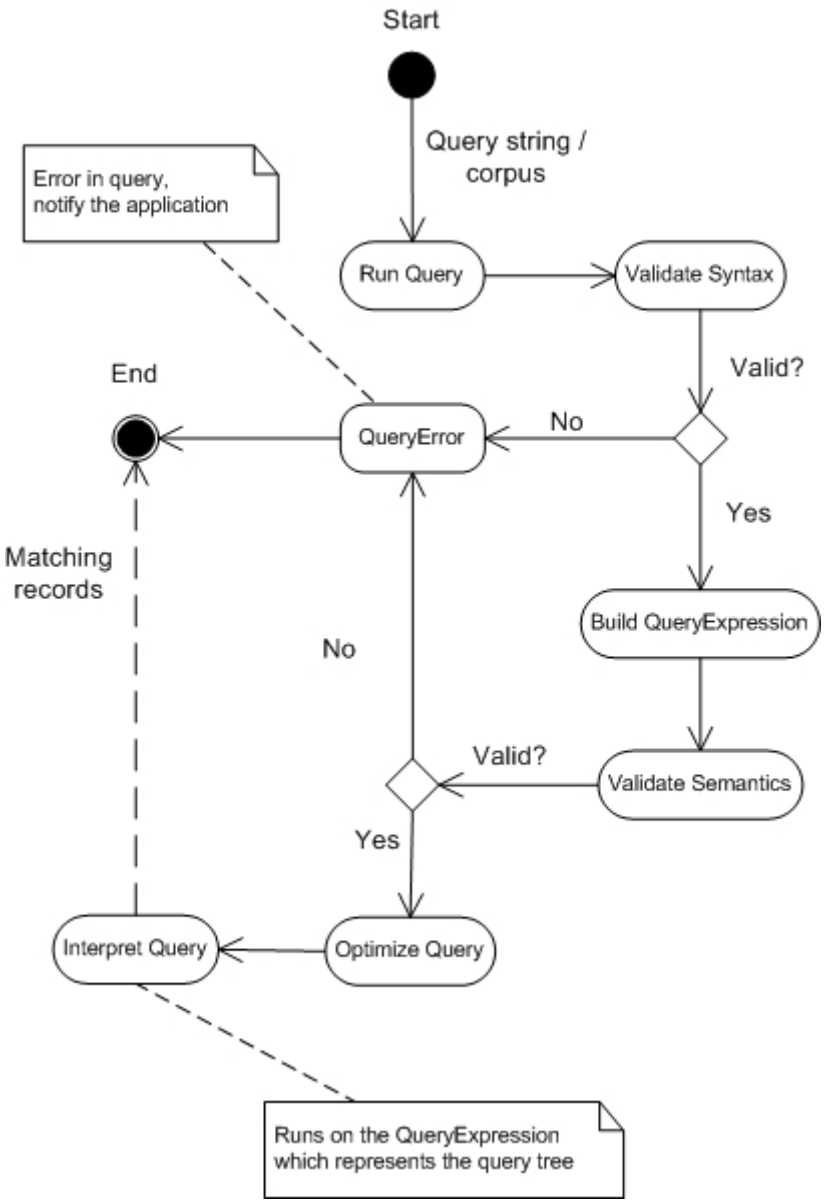


Figure 6.10 Query Interpreter Activity/State diagram

6.1.6 Reports Generation

Responsibilities and Goals

The report generation system is responsible for counting the frequencies of a given query for target attributes, noted trends, patterns within related data, or other types of reports. Statistical tendencies are drawn from frequencies and must therefore be summarized in a well structured manner.

The report generation subsystem must organize this statistical data into readable, intuitive formats or display it graphically for analysis and presentation by linguists. Exchange with other applications or human researchers must consequently be supported. This responsibility falls on the report generation module.

Responsibilities have been delegated to this system as a result of identification of the essential tasks from the following use cases:

- ReportFrequencyCounts
- GenerateStatsReport

Report generation has not been implemented in the prototype software. Consequently, no class diagram has been developed.

6.1.7 User Management

Responsibilities and Goals

Multiple user access requires administration of system data, allocation and revocation of access privileges, and user validation. User management is responsible for accepting identification from users and authenticating this information against the database.

This subsystem must also control access to only those parts of the database a user has rights to use. As this system has been designed to manage all information pertaining to users, control of users' rights is also part of user management. When users request information from the system, the user management subsystem must validate the user's identity and determine the actions that user may perform.

These responsibilities stem from the 'AuthenticateUser' use case.

Class Diagram

To authenticate a user, the system uses the UserData object defined in sub-section 6.1.1: *Data Corpus*. From this it determines the identity of the user supplying a username and password. This simple structure is given below.

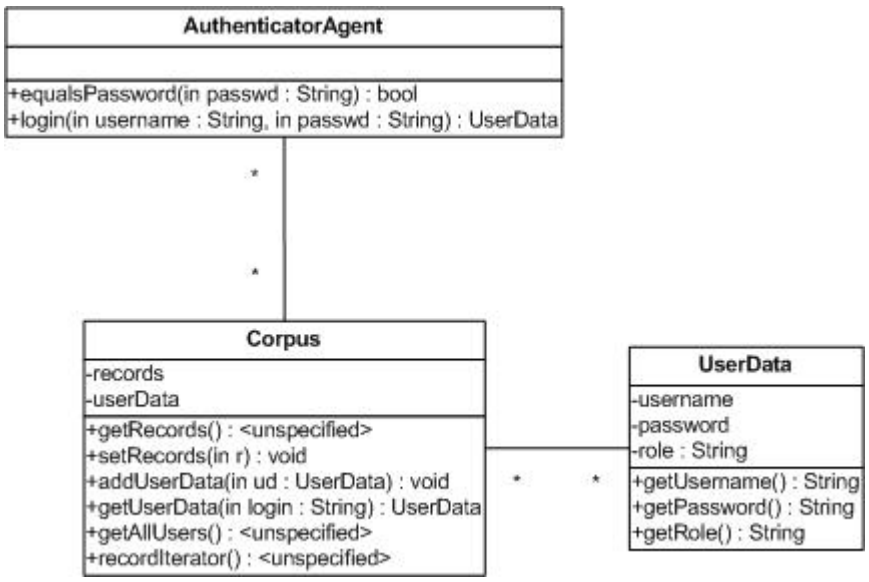


Figure 6.11 Authentication class diagram. Logically, it contains only a single class, as *Corpus* and *UserData* are taken from the Data Corpus package.

6.1.8 Graphical User Interface

Responsibilities and Goals

This subsystem provides a system interface to the user allowing them to invoke the tasks of the various modules detailed above. It is built on top of the other modules.

The graphical user interface (GUI) allows linguists to create transcripts from media files and metadata, as detailed in Section 6.1.3: *Transcript Creation* above. While this system controls the actual creation and storage of transcripts, the GUI allows a user to provide the necessary information to the system for this creation.

Similarly, the GUI allows linguists to augment transcript data. Invocation is done by the user interfacing with the system, while the Transcript Augmentation system above controls the migration of data throughout the corpus.

Queries are compiled by graphically selecting components within the GUI that specify parameters for the query interpreter system. Compilation of a syntactically correct query from items selected by a user is the responsibility of the graphical user interface subsystem.

Once queries are built and invoked, the results must be displayed to a user. The GUI system is responsible for projecting query results to the user. Users can then browse these results, make further queries, are indicate the need to generate a report.

This module is responsible for presenting a view of reports generated. Once available, reports are displayed to users in a human-readable format. The GUI subsystem does not create any new functionality within the system, but only builds a user-friendly interface to functionality provided by other subsystems.

All the use cases discussed in this paper are covered by this system indirectly. It adds a graphical component to the architecture which is built on the functionality of other modules, which directly stem from the use cases.

Class Diagram

The GUI, and indeed the entire system, is intended to be modular. Tools and subsystems can be added without affecting the functionality of the remainder of the system. This is achieved by designing each module to specify its own UI and functionality. The main application window then has handles on each of the modules, which can be invoked using the *CommonModuleInterface* methods shown in Figure 6.12. The *SystemTools* include a controller for managing and browsing corpora, a login manager, and user-preferences management.

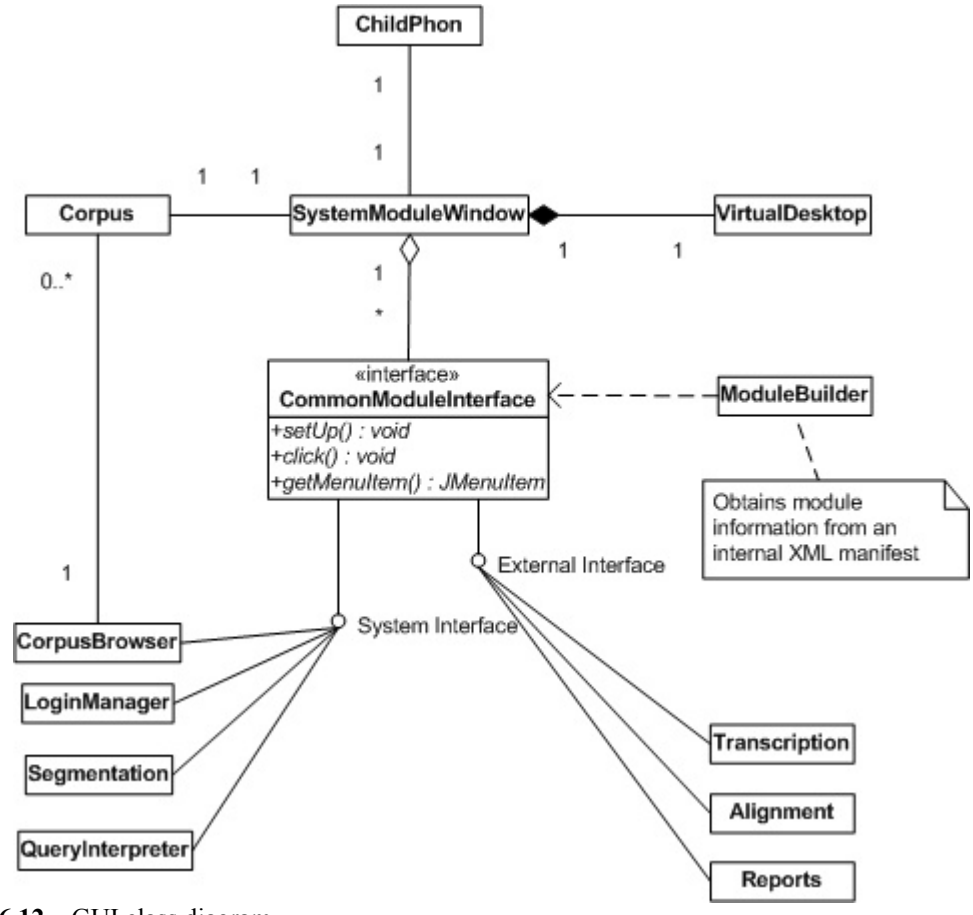


Figure 6.12 GUI class diagram

On start-up, the system discovers any modules of the system by consulting an internal manifest file. This manifest stores the location of each module and the root UI interface to run, which then builds itself (as all functionality should remain internal to that module. The module implements a common interface that details how the module is build and constructed (`void setup( )` in *CommonModuleInterface*). It also provides a *JMenuItem* for the root application.

This design fosters easy addition of modules, as long as these modules can construct themselves when asked, instantiate it when called, and provide a UI starting point for the module, by way of the *JMenutem*.

Chapter 7: Summary and Future Work

This chapter begins by summarizing the work presented in this thesis. We will discuss whether the proposed software design meets the requirements of the problem specification. This chapter will then conclude with an overview of some future development as well as some extensions to the application.

7.1 Summary

The main goal of this project was to design an application to aid linguists in language acquisition analysis. The previous chapters have defined a system that

- includes support for multi-media – the segmentation module of the system provides an interface for delimiting interesting time-slots within a given media file.
- supports Unicode – the IPA font family used supports a common character-code mapping and a uniform font rendering across platforms and applications.
- uses XML as a data storage format – extends an existing scheme to encourage sharing data with other linguistic software packages, and promote compatibility.
- provides user interfaces for segmentation, transcription, and syllable analysis – the transcription module details this design.
- includes automatic syllabification of transcribed phrases – non-destructive break down of phrases into syllables.
- includes automatic alignment of syllables – a tool not found in any other linguistic software.

- ❖ will support multiple operating systems using the Java programming language.

The defined application will greatly assist linguists with their data analysis. It offers a base system for linguistic processes with the ability to add future analysis possibilities in the future. As well, this program should replace the current *ChildPhon* application for child language analysis. The tools provided within this software will offer several advantages to linguistic research in language acquisition.

7.2 Future Work

In any software project, there are improvements that can be made. We foresee the following as possible extensions and modifications to the application.

- ❖ An agreement on a XML storage standard for incorporating the existing scheme with our proposed additions, as well as future additions.
- ❖ Implementation of modules to support phonetic, morphological and syntactic research.
- ❖ Further query language specification to implement a full-featured scripting language for the application. A "power-user" scripting language, and query optimization are among these additions.
- ❖ Modifications to the syllabification algorithm. Currently, the syllabification algorithm only supports English. In future revisions, more language specific rules can be added.
- ❖ Modifications to the alignment algorithm. The similarity function can be tweaked so that the percentage of accurate alignments is better. Also, a number of alignment algorithms could be used. This would allow the linguist to choose alignments which suit their research.
- ❖ Additional report generation. Once new modules are added to the system, different types of graphs may be necessary in order to represent the new form of

data. As well, it may be determined that different abstractions of data are necessary.

This work can be incorporated into our design with little effort. We will be working on revising the prototype software in the coming months and implementing a fully functional system sporting all the tools detailed within this thesis. We will also be making additions to the software, such as those mentioned above.

The long term goal is to have a fully functional software system ready for distribution in the near future (4-5 months) to aid in linguistic study. This, we hope, will be a beginning in an era of development in the field of linguistic software engineering.

Appendix A Sample XML File

```
<?xml encoding="UTF-8" version="1.0" ?>
<!--
    XML database sample
    Hedlund & O'Brien, 2004
    Date: 2004-04-16 18:25:34 $

    This XML file was composed to reflect the format of the Talkbank
    schema combined with our proposed additions to this scheme.

    The file contains 3 records generated by using an existing CLAN
    file and augmenting the data with experimental phonological data.

    The root element of this document is the CHAT element.  This
    denotes the origin of the information.  Some of the attributes of
    this element indicate the location of the xsi (XML Schema Instance)
    as well as the xsd (XML Schema Document); the latter can be used to
    validate the "contract" of abiding by the schema.

-->

<CHAT xsi:schemaLocation="/home/pobrien/talkbank
        /home/pobrien/newTalkbank.xsd" Version="1.0" Lang="en">

<!-- ********************* Experimental Data ********************** -->

    <!--
        Metadata
            The metadata is used to denote session information.  This
            is used as the observational data in the software system.
    -->

    <Metadata>
      <Date>2004-03-12</Date>
      <Subject>annie</Subject>
    </Metadata>

    <!--
        Comment
            Comments can specify the individual that did the
            segmentation on this file (Coder).  General comments on the
            experiment can be added such as a description of the
            experiment's setting.
    -->
```

```
    <Comment type="Coder">yrose</Comment>
    <Comment type="Generic">During this session the participant, Annie,
        is playing with toys in her playroom.  Annie's dad sits and
        plays with her.
    </Comment>

    <!--
        Participants
            Beings included in the experiment.  This can be
            transcribers, aligners, administrations, or experiment
            participants (i.e. children).
    -->

    <Participants>
        <participant id="yrose" role="admin" name="Yvan Rose" />
        <participant id="ghedlund" role="trans" name="Greg Hedlund" />
        <participant id="annie" role="participant" name="Annie May"
            sex="F" birthday="2002-10-14" language="en" />
    </Participants>

<!-- ******************* Database Records Begin ****************** -->

    <!-- first phrase of an utterance -->
    <u who="annie" type="phon">
        <ortho type="spontaneous">I hit the ball</ortho>

        <!-- target and actual phrases -->
        <phrase type="target">
            <transcript who="ghedlund">
                <ipa>aj hit</ipa>
            </transcript>
            <transcript who="yrose">
                <ipa>aj hɪt</ipa>
            </transcript>

            <!—describes the syllabification for a transcription -->
            <syllabification who="yrose">
                <syllable syl="1">
                    <constituent type="n">
                        <ipa>aj</ipa>
                    </constituent>
                </syllable>
                <syllable syl="2">
                    <constituent type="o">
                        <ipa>h</ipa>
                    </constituent>
                    <constituent type="n">
                        <ipa>ɪ</ipa>
                    </constituent>
                    <constituent type="c">
                        <ipa>t</ipa>
                    </constituent>
                </syllable>
            </syllabification>
```

```
    </phrase>

    <phrase type="actual">
        <transcript who="ghedlund">
            <ipa>aj jʰɪt</ipa>
        </transcript>
        <transcript who="yrose">
            <ipa>aj jɪt</ipa>
        </transcript>

        <syllabification who="ghedlund">
            <syllable syl="1">
                <constituent type="n">
                    <ipa>aj</ipa>
                </constituent>
            </syllable>
            <syllable syl="2">
                <!-- the onset-nucleus-coda breakdown -->
                <constituent type="o">
                    <ipa>jʰ</ipa>
                </constituent>
                <constituent type="n">
                    <ipa>ɪ</ipa>
                </constituent>
                <constituent type="c">
                    <ipa>t</ipa>
                </constituent>
            </syllable>
        </syllabification>
    </phrase>

    <!--
        Describes what syllables should be aligned between the
        target and actual transcriptions
    -->
    <alignment>
        <align targetID="1" actualID="1" />
        <align targetID="2" actualID="2" />
    </alignment>

    <media start="4.015" end="7.561" unit="s" />
</u>

<!-- Second phrase of the same utterance -->
<u who="annie" type="phon">
    <ortho type="spontaneous">I hit the ball</ortho>

    <phrase type="target">
        <transcript who="ghedlund">
            <ipa>ðə bal</ipa>
        </transcript>
        <transcript who="yrose">
            <ipa>ðə bal</ipa>
        </transcript>
```

```
    <syllabification who="yrose">
        <syllable syl="1">
            <constituent type="o">
                <ipa>ð</ipa>
            </constituent>
            <constituent type="n">
                <ipa>ə</ipa>
            </constituent>
        </syllable>
        <syllable syl="2">
            <constituent type="o">
                <ipa>b</ipa>
            </constituent>
            <constituent type="n">
                <ipa>a</ipa>
            </constituent>
            <constituent type="c">
                <ipa>l</ipa>
            </constituent>
        </syllable>
    </syllabification>
</phrase>

<phrase type="actual">
    <transcript who="ghedlund">
        <ipa>dɑb</ipa>
    </transcript>
    <transcript who="yrose">
        <ipa>dæb</ipa>
    </transcript>

    <syllabification who="ghedlund">
        <syllable syl="1">
            <constituent type="o">
                <ipa>d</ipa>
            </constituent>
            <constituent type="n">
                <ipa>ɑ</ipa>
            </constituent>
            <constituent type="c">
                <ipa>b</ipa>
            </constituent>
        </syllable>
    </syllabification>
</phrase>

 <!-- '0' denotes a null mapping
     i.e. target has two syllables while actual has only one.
     Here we align the first syllables in each but align the
     second syllable in the target with a 'null' syllable in
     the actual.
-->
<alignment>
    <align targetID="1" actualID="1" />
```

```
            <align targetID="2" actualID="0" />
        </alignment>

        <media start="4.015" end="7.561" unit="s" />
    </u>

    <!-- a morphological record -->
    <u who="annie" type="morpho">
        <w>Annie?</w>
        <pause fluency="fluent" symbolic-length="simple" />
        <w>what</w>
        <w>is</w>
        <w>this?</w>

        <a type="comments">Annie's dad holds up a plastic bat to Annie.
            Annie smiles and looks at the bat for a few seconds.</a>
        <t type="p">

        <media start="17.453" end="22.988" unit="s" />
    </u>

<!-- ********************** Multimedia Source ******************** -->

    <!-- the media source for this file (experiment) -->
    <media type="video" href="/home/pobrien/corpus110257.xml"
            start="0.0" end="542.937" unit="s" />

</CHAT>
```

Appendix B Feature Set Matrix

Each IPA character has associated with it a set of features which describe that sound. For the purposes of C*hildPhon* there are thirty-two possible features that can be present in a particular IPA character. We can therefore define a Feature Set for each possible IPA character.

The following three pages displays a table of the most commonly used IPA characters and the features present in each one. The first column of the table displays the character as it is written. The second column shows the respective Unicode character encoding in hexadecimal format. The rest of the columns define the thirty-two features which are defined for *ChildPhon*. If a particular feature is present in an IPA character, there will be a '+' symbol within the respective column for that character.

The feature sets contained within the software are open to modification by the researcher. *ChildPhon* allows for editing of the Features Sets since different linguists may have distinct views on the data.

[Feature matrix goes here]

Bibliography

[1] Bernhardt, H. Barbara, and J. P. Stemberger, *Handbook of Phonological Development: From the Perspective of constraint-based Nonlinear Phonology*. San Diego, CA: Academic Press, 1998.

[2] S. Bird, P. Buneman, and W. C. Tan, "Towards A Query Language for Annotation Graphs" in *Proceedings of the Second International Conference on Language Resources and Evaluation*, pp. 807-814, Paris, European Language Resources, 2000. Association Web: http://arxiv.org/abs/cs/0007023.

[3] D. Brown, and D. A. Watt, *Programming Language Processors in Java: Compilers and Interpreters.* New York, NY: Prentice Hall, 2000.

[4] B. Bruegge, and A. H. Dutoit, *Object-Oriented Software Engineering: Conquering Complex and Changing Systems*. Upper Saddle River, NJ: Prentice Hall PTR, 2000.

[5] S. Cassidy, "Compiling Multi-Tiered Speech Databases into the Relational Model: Experiments with the Emu System," Macquarie University, 2000. Association Web: http://emu.sourceforge.net/eurospeech99.shtml.

[6] U. Chaudhri, and L. Djeraba, (Eds.)., *XML- Based Data Management and Multimedia Engineering – EDBT 2002 Workshops*. New York, NY: Springer-Verlag Berlin Heidelberg, 2002.

[7] Cormen, Leiserson, Rivest, and Stein, *Introduction to Algorithms Second Ed.*, Cambridge, Massachusetts: MIT Press, 2001.

[8] "Eagles_on_Line." http://www.ilc.cnr.it/EAGLES96/home.html, 2004.

[9] S. E. Eddy, *XML In Plain English*. Foster City, CA: IDG Books Worldwide, Inc., 1998.

[10] H. J. Giegerich, *English Phonology: An Introduction*. Cambridge: Cambridge University Press, 1992.

[11] H. Gord and Y. Rose, Input Elaboration, Head Faithfulness and Evidence for Representation in the Acquisition of Left-edge Clusters in West Germanic. In R. Kager, J. Pater and W. Zonneveld (eds.). Constraints in Phonological Acquistion. Cambridge, Cambridge University Press, 109-157, 2003.

[12] D. Gusfield, *Algorithms on Strings, Trees, and Sequences*. New York, NY: Cambridge University Press, 1997.

[13] C. Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Anaylsis and Design and the Unified Process*. Upper Saddle River, NJ: Prentice Hall PTR, 2002.

[14] Y. Rose, Headedness and Prosodic Licensing in the L1 Acquistion of Phonology. Ph. D. Dissertation, McGill University. 2000.

[15] Y. Rose, "ChildPhon: A Database Solution for the study of Child Phonology." Memorial University of Newfoundland: Department of Linguistics, 2002.

[16] SUN Microsystems Inc. "Java Developer Home." http://developers.sun.com/index.html, 2004.

[17] "XML for Talkbank." http://xml.talkbank.org/, 2004.

[18] World Wide Web Consortium. http://www.w3c.org/, 2004

[19] W. M. Zuberek, Notes: Programming Languages and Their Processors. Memorial University of Newfoundland: Department of Computer Science, 2002.

[20] Object Management Group, Inc. UML Specification. 2004. Association Web: http://www.uml.org/

[21] E. O. Selkin, Prosodic Domains in Phonology: Sanskrit Revisited. In M. Aronoff and M. Kean (eds.). Juncture. A Collection of Original Papers. Saratoga, CA: Anma Libri. 107-129. 1980