Académie de Paris
Université Paris 6 – Pierre et Marie Curie
École Doctorale d'Informatique

THÈSE DE DOCTORAT

spécialité
ACOUSTIQUE, TRAITEMENT DE SIGNAL ET INFORMATIQUE
APPLIQUÉS À LA MUSIQUE

présentée par
Diemo Schwarz

pour obtenir le grade de
DOCTEUR de l'UNIVERSITÉ PARIS 6 – PIERRE ET MARIE CURIE

DATA-DRIVEN CONCATENATIVE SOUND SYNTHESIS

soutenue le 23. 1. 2004

devant le jury composé de

XAVIER RODET	Directeur de Thèse
UDO ZÖLZER	Rapporteur
FRANÇOIS PACHET	Rapporteur
THIERRY DUTOIT	Examinateur
GÉRARD CHOLLET	Examinateur
JEAN-FRANÇOIS PERROT	Examinateur

PhD Thesis in
Acoustics, Computer Science, Signal Processing
Applied to Music

Data-Driven Concatenative Sound Synthesis

Diemo Schwarz
schwarz@ircam.fr
http://www.ircam.fr/anasyn/schwarz

Version 1.01
19th January 2004

Ircam – Centre Pompidou
1, place Igor–Stravinsky
75004 Paris, France
http://www.ircam.fr

University of Paris 6 – Pierre et Marie Curie

. . . à Marie-Ève

Chapter Overview

Abstract

Concatenative data-driven sound synthesis methods use a large database of source sounds, segmented into heterogeneous *units*, and a *unit selection* algorithm that finds the units that match best the sound or musical phrase to be synthesised, called the *target*. The selection is performed according to the features of the units. These are characteristics extracted from the source sounds, e.g. pitch, or attributed to them, e.g. instrument class. The selected units are then transformed to fully match the target specification, and concatenated. However, if the database is sufficiently large, the probability is high that a matching unit will be found, so the need to apply transformations is reduced.

Usual synthesis methods are based on a model of the sound signal. It is very difficult to build a model that would preserve all the fine details of sound. Concatenative synthesis achieves this by using actual recordings. This data-driven approach (as opposed to a rule-based approach) takes advantage of the information contained in the many sound recordings. For example, very naturally sounding transitions can be synthesized, since unit selection is aware of the context of the database units.

In speech synthesis, concatenative synthesis methods are the most widely used. They resulted in a considerable gain of naturalness and intelligibility. Results in other fields, for instance speech recognition, confirm the general superiority of data-driven approaches. Concatenative data-driven approaches have made their way into some musical synthesis applications which are briefly presented.

The CATERPILLAR software system developed in this thesis allows data-driven musical sound synthesis from a large database. However, musical creation is an artistic activity and thus not based on clearly definable criteria, like in speech synthesis. That's why a flexible, interactive use of the system allows composers to obtain new sounds.

To constitute a unit database, alignment of music to a score is used to segment musical instrument recordings. It is based on spectral peak structure matching and the two approaches using Dynamic Time Warping and Hidden Markov Models are compared.

Descriptor extraction analyses the sounds for their signal, spectral, harmonic, and perceptive characteristics, and temporal modeling techniques characterise the temporal evolution of the units uniformly. However, it is possible to attribute score information like playing style, or arbitrary information to the units, which can later be used for selection.

The database is implemented using a relational SQL database management system for optimal flexibility and reliability. A database interface cleanly separates the synthesis system from the database.

The best matching sequence of units is found by a Viterbi unit selection algorithm. To incorporate a more flexible specification of the resulting sequence of units, the constraint solving algorithm of adaptive local search has been alternatively applied to unit selection. Both algorithms are based on two distance functions: the target distance expresses the similarity of a target unit to the database units, and the concatenation distance the quality of the join of two database units.

Data-driven concatenative synthesis is then applied to instrument synthesis with high level control, explorative free synthesis from arbitrary sound databases, resynthesis of a recording with sounds from the database, and artistic speech synthesis. For these applications, unit corpora of violin sounds, environmental noises, and speech have been built.

Résumé

La synthèse concaténative par sélection d'unités sonores utilise une base de données de sons enregistrés, et un algorithme de *sélection d'unités* qui choisit les segments de la base de données qui conviennent le mieux pour la séquence musicale que l'on souhaite synthétiser, dite la *cible*. Ensuite, ces segments sont concaténés pour former la phrase cible. La sélection est fondée sur les caractéristiques de l'enregistrement, qui sont obtenues par analyse du signal et correspondent par exemple à la hauteur, à l'énergie ou au spectre.

Les méthodes de synthèse musicale habituelles sont fondées sur un modèle du signal sonore, mais il est très difficile d'établir un modèle qui préserverait la totalité des détails du son. En revanche, la synthèse concaténative, qui utilise des enregistrements réels préserve ces détails.

Si la base de données des enregistrements est suffisament grande, on dispose d'une grande quantité de sons dans de nombreux contextes, ce qui permet de minimiser l'application de transformations, qui entraînent toujours une dégradation du son. Cette approche, dite *approche fondée sur les données*, bénéficie des informations contenues dans les nombreux enregistrements sonores. Au contraire, l'approche dite "fondée sur les règles" construit les règles de façon réflexive, ce qui peut être source d'erreurs.

En synthèse de la parole, les méthodes de synthèse concaténative sont les plus employées. Ces systèmes sont géneralement considérés comme plus performants que les systèmes de synthèse paramétriques fondés sur les règles pour le naturel et l'intelligibilité. En effet, les résultats dans d'autres domaines, comme celui de la reconnaissance de la parole, confirment la supériorité générale de l'approche fondée sur les données. Les idées de la synthèse concaténative fondée sur les données apparaissent dans d'autres applications et systèmes de synthèse musicale. Ceux-ci sont brièvement présentées et analysées.

Le système logiciel CATERPILLAR, réalisé au cours de cette thèse, permet de réaliser la synthèse sonore musicale concaténative fondée sur les données. Or, la création musicale est une activité artistique, et n'est donc pas fondée sur des critères rigoureusement définis, comme c'est le cas en synthèse de la parole, celle-ci accordant un interêt primordial à l'intelligibilité et au naturel. C'est pourquoi l'utilisation de ce système de synthèse musicale pour une activité créatrice permet également aux compositeurs et musiciens d'atteindre de nouvelles sonorités. Le système est capable d'intégrer d'autres bases de données de sons, des caractéristiques supplémentaires, et de nouveaux algorithmes de sélection. Les fonctionnalités de CATERPILLAR sont :

- l'analyse et la segmentation des enregistrements destinés à fournir le matériel source
- l'analyse en descripteurs sonores et la modelisation temporelle des unités sonores
- la gestion des fichiers de son et de données dans la base de données
- la recherche et la sélection d'unités de la base de données en fonction des paramètres cibles
- la concaténation des unités

La segmentation d'enregistrements musicaux est obtenue par alignement de la partition avec le signal sonore d'une exécution de celle-ci. L'alignement est fondé sur une méthode d'appariement spectral, utilisable aussi dans un cas polyphonique, et deux méthodes différentes, qui sont comparés, le Dynamic Time Warping et les chaînes de Markov cachées (Hidden Markov Models).

L'analyse en descripteurs fournit des caractéristiques du signal, spectrales, harmoniques et perceptives. Toutefois, il est possible d'attribuer des informations de la partition (modes de jeu) ou des informations arbitraires aux unités. La modélisation temporelle réduit les courbes continues des descripteurs à un vecteur de caractéristiques qui décrivent l'évolution temporelle d'un descripteur à travers une unité.

La base de données est implantée dans un système de gestion de bases de données relationelles, pour une flexibilité, extensibilité et fiabilité optimisée. Un interface unique sépare la base de données du reste du système, afin qu'elle puisse être remplacée par une autre base ou un autre système de gestion.

Deux algorithmes de sélection d'unités ont été développés et sont comparés : L'un utilise la méthode classique de recherche du meilleur chemin à travers un réseaux d'états par l'algorithme de Viterbi, l'autre formule la sélection comme un problème de résolution de contraintes. Les deux sont basés sur des fonctions de distance dont la distance cible exprime la similarité d'une unité cible avec des unités de la base, et la distance de concaténation la qualité de l'enchaînement entre deux unités de la base.

La synthèse concaténative fondée sur les données est ensuite appliquée à la synthèse haut niveau d'un instrument, à la synthèse libre, sorte de généralisation de la synthèse granulaire avec un contrôle effectif du résultat sonore, à la resynthèse d'un enregistrement avec les sons de la base, et à la synthèse de la parole artistique. Pour ces applications des corpus d'unités de violon, de bruits environnementaux et de parole ont été constitués.

Acknowledgments

I'd like to thank my thesis director, Xavier Rodet, who accepted me in his team to do this work, the members of the jury, first of all my reporting examiners Udo Zölzer and François Pachet for their precious remarks and encouragements, Gerard Chollet, Jean-François Perrot for their time and expertise, and especially Thierry Dutoit whose talk about EULER at Ircam in the year 1998 inspired me to take up the subject of concatenative musical synthesis based on unit selection,

Nicola Orio, whom I had the great opportunity to join in his research and development on score following and alignment, and who inspired me very much with his scientific rigour paired with never ending generosity and humor,

Norbert Schnell, my boss, for much comprehension and motivation for finishing this thesis while working during the last one and a half years, Hugues Vinet for being understanding and flexible with my work contract,

David Ralley, Alexis Baskind, and Fabien Gouyon for the heavy task of proofreading the manuscript and their many helpful remarks,

Olivier Lartillot, Thomas Helie, Carlos Agon, and Joël Bensoam for the indispensable support between (ex-)PhD students,

Axel Röbel, Kasper Souren, Fabrice Chapuis for providing me with various well installed laptops so that I could write this document in various calm and cozy places (see below),

Arnaud Mewissen, Gérard Assayag, Laurent Worms, Bertrand Delezoïde and Eric Daubresse for providing me with rich and varied sound sources I could use to stuff out my database with,

Max Jacob for helping me hunt down and fix the performance bottlenecks showing up with a such heavily stuffed database, and for general bliss,

my colleagues at Ircam, Patrice Tisserand, Riccardo Borghesi, Frédéric Bevilacqua, Jean-Philippe Lambert, Emmanuel Vincent, Vincent Rioux, Geoffroy Peeters, Guillaume Lemaitre, Olivier Pasquet, Guillaume Boutard, and Sebastien Roux for much support, fun, and the music,

Guillaume Vandernoot for supporting loud music from my office right next to his in the long thesis evenings.

I am grateful to Sylvie Noël, Ferreol Soulez, Örsten Kärki, Thomas Foirien, who during their internships helped a lot pushing the work further.

Thanks goes to Mélanie and Thomas, and especially Isabelle and Eric for putting up with the strange character one develops while finishing a thesis.

I'd like to express my deepest gratitude to Alain Bouillon, Jacqueline Bouillon, Claudine Ybert, François Morel, Fernande Doyère, Paulette Doyère, Anne Duquesney and Bertrand Duquesney from around Coutances, Normandy, (*thèse sous le pommier*) for providing me calm and cozy shelter where I could go on writing this document undisturbed, Loriana and Salomé, Benjamin and Clemence, Arthur, Quetsch and Siam for keeping it that way,

to my parents Margret and Rudolf Schwarz, to Andreas Tichy, Gaby and Susanne Gantner, and to Marie-Eve Bouillon for her incommensurable patience(!) and much more support than anyone could ever ask for.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

When technology advances and is easily accessible, creation progresses, too, driven by the new possibilities that are open to be explored. For musical creation, we have seen such surges of creativity throughout history for example with the first easily usable recording devices, the phonograph and magnetic tape recorders, in the 1940s, with widespread diffusion of electronic synthesizers at the end of the 1970s and beginning of the 1980s, and with the availability of real-time interactive digital processing tools at the end of the 1990s.

The next relevant technology advance is already here, widespread diffusion just around the corner, and waiting to be exploited for creative use: Large databases of sound, with a pertinent description of their contents, ready for content-based retrieval. These databases want to be exploited for musical sound synthesis.

This work proposes data-driven concatenative musical sound synthesis from large heterogeneous sound databases. Musical sound synthesis allows the creation of new sounds, either from scratch, or by changing an existing sound (this is usually called resynthesis). In both cases, the parameters of the synthesis model used have to be specified. In synthesis from scratch they are completely given by the user. In resynthesis, the parameters obtained by analysing an existing sound are modified.

It can be hard to specify the right synthesis parameters in order to obtain a desired result. Often it can not be feasibly done manually, so that the specification of these parameters is done by defining *rules*. However, setting up the rules is error prone, since they can be very complex and numerous. This is where a *data-driven* approach is needed to exploit the information contained in the data, and improve the control of the synthesis.

Concerning the synthesis result, there are certain effects in the sound which are hard to model in sufficient quality and precision, e.g. transients from musical instruments. Here, we need *concatenative* synthesis, because, by its use of actual recordings, the totality of the fine details of the sound are preserved.

Putting both together gives us concatenative data-driven sound synthesis. This method uses a large heterogeneous database of source sounds — either sound snippets, single notes, or complete phrases. The source sounds are time-segmented into *units* and analysed for their sound *descriptors*. These are characteristics extracted from the source sounds, e.g. pitch, or attributed to them, e.g. instrument class.

A *unit selection* algorithm finds the units that match best the sound or phrase to be synthesised, called the *target*, expressed also in descriptors. The selected units are then transformed to fully match the target specification, and concatenated. However, if the database is sufficiently large, the probability is high that a matching unit will be found, because a large number of sound events in many different contexts are available, so the need to apply transformations, which always degrade the sound, is reduced.

The data-driven approach takes advantage of the information in the many sound examples in synthesis by supplying the fine details from the database while the rough characteristics of the desired sound are given by the target specification. Instead of supplying rules constructed by careful thinking, the rules are induced from the data itself. This hypothesis is illustrated in figure 1.1, where the relations between the score and the produced sound in the case of performing an instrument, and the synthesis target and the unit descriptors in the case of concatenative data-driven synthesis are shown on their respective level of representation of musical information (according to Vinet (2003), we can classify digital musical representations into the physical level, the signal level, the symbolic level, and the knowledge level).

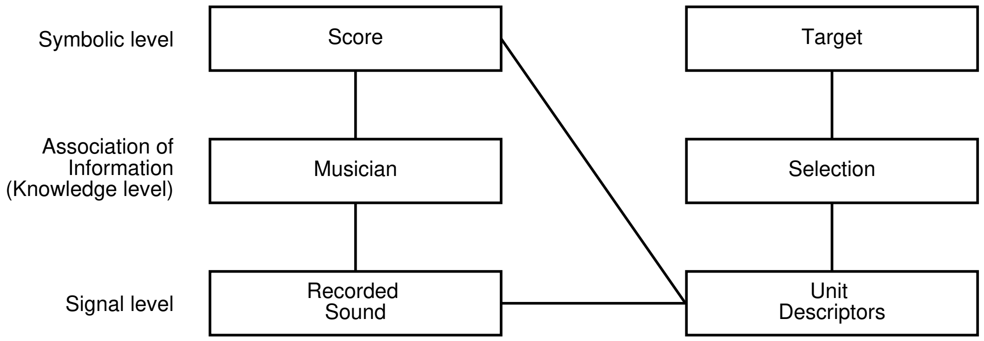


Figure 1.1: Hypothesis of high level synthesis

Research in musical synthesis is heavily influenced by research in speech synthesis, which can be said to be roughly 10 years ahead. Concatenative unit selection speech synthesis from large databases is used in a great number of Text-to-Speech systems for waveform generation. Its introduction resulted in a considerable gain in quality of the synthesized speech. Unit selection algorithms attempt to estimate the appropriateness of a particular database speech unit using linguistic features predicted from a given text to be synthesized. The units can be of any length (non-uniform unit selection), from sub-phonemes to whole phrases, and are not limited to diphones or triphones. Those data-driven speech synthesis systems are generally considered superior to rule-based parametric synthesis systems in terms of naturalness and intelligibility.

Despite its promising approach and its success in speech synthesis systems, concatenative data-driven methods have been, up to recently, rarely used in musical synthesis. Although concatenative data-driven sound synthesis is quite similar to concatenative speech synthesis and shares many concepts and methods, both have different goals. Even from a very rough comparison between musical and speech synthesis, some profound differences spring to mind, which make the application of concatenative data-driven synthesis techniques from speech to music non-trivial:

- Speech is a-priori clustered into phonemes. A musical analogue for this *phonemic identity* are pitch classes which are applicable for tonal music, but in general, no a-priori clustering can be presupposed.
- In speech, the time position of synthesized units is intrinsically given by the required duration of the selected units. In music, precise time-points have to be hit when we want to keep the rhythm.
- In speech synthesis, intelligibility and naturalness are of prime interest, and the synthesised speech is often limited to "normal" informative mode. However, musical creation is based on artistic principles, uses many modes of expressivity, and needs to experiment. Therefore, creative and interactive use of the system should be possible by using any database of sounds, any features, and a flexible expression of the target for the selection.

Data-driven synthesis is now more feasible than ever with the arrival of large sound database schemes, e.g. in the European and international projects detailed in chapter 2.2. They finally promise to provide large sound corpora in standardised description. It is this constellation that provided the basis for great advancements in speech research: the existence of large speech databases allowed corpus-based linguistics to enhance linguistic knowledge and the performance of speech tools.

As seems to happen often when researching a new synthesis method,[1] we discover that, first of all,

[1] Last remarked by Peeters (2001) for PSOLA synthesis.

the corresponding analysis methods have to be researched or refined. In the case of data-driven concatenative synthesis, we depend heavily on analysis in two aspects:

For the data-driven part, the signal analysis for descriptors is fortunately grounded on long and stable research. In the context of national, European and international research projects, it has recently been developed further for the purpose of audio content description, which is exactly what we need. The only thing that needed to be developed was the characterisation of the continuous descriptor data per synthesis unit.

For the concatenative part, however, the analysis needed is the precise segmentation of audio into units. This is standard for speech, but has not been addressed in the required precision for music. That's why a large amount of work has been devoted to music alignment, using the two different methods of dynamic time warping and hidden Markov models.

These alignment methods are used to segment the source sounds that fill the database of the CATERPILLAR software system that has been developed to perform concatenative musical sound synthesis. A fairly large corpus of sound data (instrument sounds, voices, environmental and electronic noises) has been segmented, analysed for their descriptors, modeled for their temporal evolution, and stored in an SQL database.

CATERPILLAR uses the classical unit selection algorithm from speech synthesis based on Viterbi path finding. However, to fulfill the above-mentioned goal of flexibility for musical creation, a different formalism was needed to easily express more requirements for the synthesis target. This formalism is found in the research on constraint satisfaction.

1.1 A Note on Terminology

Some central notions used throughout this document can have different meanings or synonyms, which are clarified or at least highlighted here.

A *unit* is a temporal *segment* in a sound file plus the data describing it, i.e. the characteristic values of its descriptors. A source unit always lives in the CATERPILLAR database, a target unit can be persistent, too, or exist temporarily just for selection. In the context of synthesis, unit can also refer to the chunk of sound signal contained in the segment.

A *descriptor* describes a certain quality of a sound, which can evolve over time. This temporal evolution is modeled in the *characteristic values* of a unit. Descriptor is often used interchangeably with *feature*. In the narrower sense, each characteristic value of a descriptor is a feature. Note the possible confusion between a *descriptor type* (e.g. pitch), and the *descriptor data* (the pitch curve of a specific unit), which might both be referred to as descriptor.

Given here for comparison, in the context of the MPEG-7 multimedia description standard (see section 2.2.1), the definition of Hunter (1999) draws a different distinction between descriptor and feature on the level of value vs. representation:

> *A descriptor defines the syntax and the semantics of one representation of a particular feature of audiovisual content. A feature is a distinctive characteristic of the data which is of significance to a user.*
>
> *For example, the* color *of an image is a feature. Possible* Descriptors *corresponding to the* color *feature are: color histogram, RGB vector or a string. A Descriptor value is an instantiation of a Descriptor for a given data set. For example, RGB = (255, 255, 255), colorstring = "red".*

A *phoneme* is a class of *phones*. Or, the other way round, a phone is an instance (an actual occurrence in a speech signal) of a phoneme (a linguistic object). For *diphones*, a speech segment encompassing the last half of a phone and first half of the next phone including the transition, this distinction is less widespread. Diphone is often used for the instance and the class, which should be called *diphoneme*. The two halves of a phone or a diphone are called *semiphones*.

For a musical *note* there is no habitual distinction at all between an instance, i.e. a note occurring in a score or signal of a certain pitch, and the class of all notes with that pitch. There is also not yet a music-specific term for the concept of diphone, i.e. a segment that goes from the middle of a note's sustained part through to the middle of the next note. In this work, we will refer to is as *dinote*, but there are some established uses of the term diphone, e.g. in the DIPHONE program (see section 18.6.2), which is appropriate since *phone* means simply *sound*.

Finally, there is a possible confusion for the term *database* between the database management system ("a query sent to the database"), the database schema ("the database stores references to sound files"), and its contents ("a database of violin sounds"). The latter is better called *corpus*.

1.2 Outline of this Document

Part I describes work previous and related to concatenative data-driven synthesis, among which an overview of other research domains using data-driven approaches in chapter 2, followed by a large chapter 3 on speech synthesis, where the history of the development of concatenative speech synthesis is summarised, and the classical unit selection algorithm by searching a best path through a state transition network is explained. Singing voice and musical synthesis is treated in chapter 4. Finally, chapter 2.2 gives an account of national, European, and international research projects related to data-driven synthesis.

Part II describes the work on music alignment: After an introduction to alignment explaining the principle, applications, and requirements, we delve deeply into two different methods: alignment by dynamic time warping in chapter 6 and alignment with hidden Markov models in chapter 7. Chapter 8 compares the two approaches and presents results.

Part III describes the CATERPILLAR system for concatenative data-driven synthesis: After an overview in chapter 9, we explain the calculation of the sound descriptors in chapter 10, and the characterisation of the continuous descriptor data per synthesis unit in chapter 11.

The following four chapters explain the database developed for CATERPILLAR: The basics of relational databases and modeling are described in chapter 12, and the database interface and the SDIF Sound Description Interchange Format, which is used for all import of data, in chapter 13. The actual CATERPILLAR database architecture is explained in chapter 14, and the data in it in chapter 15, describing the contents of some corpora used for synthesis.

The synthesis part is explained in chapter 16, with details of the distance functions used, the explanation of the concept of preselection, and the presentation and discussion of two different approaches to the unit selection algorithm (one by best path search as in speech synthesis, one as a constraint satisfaction problem), and finally transformation and concatenation.

Chapter 17 describes many applications of data-driven concatenative synthesis: high level instrument synthesis, resynthesis of audio, loop based synthesis, free synthesis from arbitrary sound databases, and artistic speech synthesis. Finally, we present a conclusion and possible directions for future work in chapter 18.

The appendices document the CATERPILLAR SQL database schema in appendix A, the APIs of the database interface in appendix B, of the graphical database explorer in appendix C, and the scripts to generate this documentation in appendix D. The SDIF description types defined for this work are detailed in appendix E. For the artistic speech synthesis application, the X-SAMPA phonetic alphabet is documented in appendix F and the hierarchies of the defined phonetic categories in appendix G.

1.3 The Name of the Game[2]

Why is the name of the system CATERPILLAR? Because it needs one — "our new data-driven concatenative synthesis system based on unit selection" is just too long. CATERPILLAR has con*cat*enation

[2] This section is named in homage to Donald E. Knuth without whom generations of scientists could not have published their findings with the beauty and ease made possible by TEX and METAFONT (Knuth 1990).

in it, and caterpillar, the insect, with the segments making up its body, is a good symbol for a sequence of concatenated units, besides being cute. One might additionally remark that it traverses *trees*, *selecting* the leaves that best *match* its appetite, just as CATERPILLAR, the software system (see figures 10.3ff for some preferred specimen of trees of the latter).

The verb *concatenate* is from Latin *catena*, "chain", abbreviated to *cat* in the Unix shell command, found in caterpillar, the insect, and seen in the chains of the Caterpillar$^{\mathrm{TM}}$ building machines.

Part I

Previous and Related Work

Un jour, on m'a raconté cette histoire d'une conférence scientifique en Inde, où un astrophysicien venait de parler du Big Bang et de l'histoire de l'univers. Un vieux sage s'approche et lui dit : "Ce n'est pas comme ça que ça se passe. Selon nos livres saints, l'univers repose sur la carapace d'une tortue". L'astrophysicien, qui ne s'en laisse pas conter, lui rétorque : "Oui, mais cette tortue, elle repose sur quoi ?" Le vieil homme indique qu'elle repose "Sur la carapace d'une autre tortue". Et ajoute, plaisantant le scientifique : "Vous, je vous voir venir. Ne jouez pas à l'intelligent. Des tortues, il y en a jusqu'en bas !"

Hubert Reeves

Chapter 2

Overview

Data-driven approaches are used in many fields, examples of which are given in this and the next chapters of Part I. As many methods in musical synthesis, concatenative sound synthesis is inspired by research and development in speech synthesis. Chapter 3 gives an overview of the speech synthesis methods in use today and in a historical context, details research and development in concatenative speech synthesis, and presents the classical path-search unit selection algorithm. Chapter 4 relates concatenative sound synthesis to other methods, techniques, and artistic intentions in use in musical sound synthesis, and to singing voice synthesis, situated between sound and speech synthesis. Finally, section 2.2 lists recent national, European and international research projects with interesting links to data-driven synthesis.

2.1 Other Fields Using Data-Driven Approaches

Many findings in domains other than text-to-speech synthesis or musical sound synthesis corroborate the general superiority of data-driven approaches. Some non-exhaustive examples are given in the following. For the remaining chapters of Part I on previous and related work, we will then only consider methods that are data-driven *and* concatenative.

2.1.1 Computational Linguistics

For speech recognition, Hermansky (1998, 1999) notes that the speech data contains many important hints for recognition and exploits these by adapting filters from a large amount of data, greatly improving recognition rate — the so-called *RASTA filters* (Hermansky, Hanson, and Wakita 1985; Avendano, van Vuuren, and Hermansky 1996; van Vuuren and Hermansky 1997).

Speech synthesis of rare languages has been made possible by data-driven methods. Black and Llitjós (2002) describe that for languages where the phoneme set is not known, this phonetic knowledge can be replaced by acoustics and textual contexts, plus higher level information. That means that even the last remnant of rule-based approaches, the letter-to-sound rules, can be thrown out of text-to-speech synthesis.

Recently, data-driven unit selection methods have also been applied to concatenative *visual speech synthesis*, where the image of a talking head is generated from a database of recorded video sequences. Huang, Cosatto, and Graf (2002) use triphones as the basic unit, achieving real-time performance while still generating photo-realistic images. This was impossible to achieve before with parametrically rendered images.

Another recently developed use of data-driven methods is for the transmission of speech at very low bitrates: Segmental coding (Černocký, Baudoin, and Chollet 1998; Černocký, Baudoin, Petrovska-Delacrétaz, Hennebert, and Chollet 1998; Kopeček, Baudoin, and Chollet 1999; Baudoin, Černocký, Chollet, and Gournay 2000; Baudoin, Capman, Černocký, Chami, Charbit, Chollet, and Petrovska-Delacrétaz 2002) skillfully combines speech recognition, language independent segmentation (Chollet, Černocký, Constantinescu, Deligne, and Bimbot 1999), classification, and unit selection. On the transmitter side, speech is automatically segmented and the segments classified according to their similarity to units from a training corpus. Only the segmental identity label and prosodic information is transmitted. The receiver performs dynamic unit selection speech synthesis from the same corpus according to the transmitted segment labels, where the concatenation quality is improved by selecting units from the correct unit context (the same preceding unit class). This way, speaker-dependent intelligible speech can be transmitted using extremely low bitrates down to 200 bits/s. Baudoin, Capman, Černocký, Chami, Charbit, Chollet, and Petrovska-Delacrétaz (2002) describe the extension of this method to speaker-independent coding.

In machine translation, a new *Rosetta stone approach* makes intelligible (although of not very high literary quality) automatic translation of text possible (Mankin 2003). The method proposed by Och and Ney (2002) and Bender, Macherey, Och, and Ney (2003) uses the alignment of huge corpora of parallel texts (150 million words), and statistical modeling of the correspondences between phrase parts to retrieve understandable translations conveying the correct meaning, without any modeling of language rules.

2.1.2 Music Information Retrieval

For content-based retrieval (CBR) in the field of music information retrieval (MIR), Foote (1999) describes the advantages of a data-driven audio retrieval method.

An interesting use for of "unit" selection is the building of audio résumés based on musical structure (Peeters and Rodet 2003b): From the descriptors calculated on a musical piece, a limited number of classes are derived. Then, for each class, a short section (of a few seconds) is selected that best represents the class. These sections are concatenated and allow thus to obtain a quick preview of the musical content of a song, without having to listen to it in its entirety.

On the analysis side, Spevak (2001) and Spevak and Favreau (2002) propose a system called *Soundspotter* whose aim was initially to aid transcriptions and analysis of acousmatic music, i.e. tape pieces of concrete music. It has been extended to perform content based retrieval of passages in audio files that are perceptually similar to a selected passage. The retrieval is done by pattern matching using an artificial neural network of type *self organising map* (SOM) to recognise occurrences of the wanted extract by training on a set of descriptors.

2.1.3 Computer Music

In music composition, David Cope's *Experiments in Musical Intelligence* (Emmy) (Cope 1996) try to capture the style of a given composer by inducing rules from many examples of his compositions (in a symbolic score representation), that are then used to generate new pieces by recombination of parts of the original music at different levels of detail. Cope (2003) underlines the advantages of data-driven vs. rule based approaches as follows:

> *My first exploration with Experiments in Musical Intelligence involved coding the rules of basic part-writing as I understood them. [...] Having an intermediary — myself — form abstract sets of rules for composition seemed artificial and unnecessarily premeditative. As well, having to code new sets of rules for each new style encountered proved daunting. I therefore revised the program to create new output from music stored in a database. My idea was that every work of music contains a set of instructions for creating different but highly related replications of itself.*

In the field of real-time score following and music alignment (see Part II), Loscos, Cano, and Bonada (1999b) remark that the use of Hidden Markov Models (HMM, see section 7) means moving from the early knowledge-based (i.e. rule-based) systems such as (Dannenberg 1984; Vercoe 1984; Puckette 1995; Baird, Blevins, and Zahler 1993) to stochastic (i.e. data-driven) models such as (Grubb and Dannenberg 1998; Raphael 1999b; Raphael 2001c; Loscos, Cano, and Bonada 1999b; Orio and Déchelle 2001; Shalev-Shwartz, Dubnov, Friedman, and Singer 2002), with a massive increase of robustness and quality. See section 7.2 for more details about score following.

Data-driven synthesis can refer to other aspects than the concatenative approaches developed in this work and described in chapter 4: In analysis–synthesis, the inversion of physical models can be data-driven, as described by D'haes and Rodet (2001, 2002, 2003). A non-data-driven approach to the same problem is the analytical inversion of the model pursued in (Hélie 2002). A partial application of data-driven methods to parametric synthesis is the learning of signal models to obtain the correct mapping from control parameters to synthesis parameters for resynthesis as described in (Wessel, Drame, and Wright 1998).

Another example of the application of data-driven methods to sound synthesis is given by Jehan and Schoner (2001a, 2001b), where an instrument signal (here a violin) can control the performance parameters of a synthesiser.

2.2 Research Projects Related to Data-Driven Synthesis

There are a number of multi-annual national, European, and international research projects related to data-driven synthesis, funded by various technology initiatives, such as the European programmes IST (*Information Society Technologies*)[1], COST (*European Cooperation in the field of Scientific and Technical Research*)[2], the French programmes PRIAMM (*Programme pour la recherche et l'innovation dans l'audiovisuel et le multimedia*)[3], and the follow-up RIAM (*Recherche et innovation en audiovisuel et multimedia*)[4] (research and innovation for the audiovisual and multimedia).

In general, they aim to bring together research institutes and commercial organisations from different countries, to link innovative technology development and generic research issues with user needs in downstream applications. They try to ensure that the emerging technologies and products are suited to the multi-cultural and multi-lingual European markets.

These projects, mainly on audio indexing and retrieval, are of multiple uses for data-driven concatenative synthesis: They define standardised high-level and signal level descriptors, and spawn research for their automatic computation and classification. All of this aims at database building, which can be used as a base for data-driven synthesis.

The further research projects described in sections 2.2.5 and 2.2.6 are related to ancillary problems encountered during this work: The search for a universal and rich music notation format. See section 5.4 for a definition of the problem and a survey of possible solutions.

2.2.1 MPEG-7 Audio

The *Moving Picture Experts Group* (MPEG) is part of the *International Standardisation Organisation* (ISO). In 1991, the group started working on a series of multimedia standards of the same name. Unlike its predecessors MPEG-1 (1991), MPEG-2 (1993) and MPEG-4 (1999), which were about coding and compression of multi-media content, MPEG-7 is about describing this content. It is described in more detail in (MPEG 2003; Hunter 1999; Thom, Purnhagen, Pfeiffer, and MPEG Audio Subgroup 1999)

The audio part of the standard defines a number of *low-level descriptors* (LLDs), that can be automatically extracted from the signal, or are content information (metadata) given by an editor. The LLDs are then grouped into *descriptor schemes* (DS's) to suit a specific application.

Version 2 of MPEG-7 Audio has just been adopted, adding some missing descriptors, and version 3 is under discussion. Some work on instrument and general sound classification in the framework of MPEG-7 Audio is described in (Peeters, McAdams, and Herrera 2000; Peeters and Rodet 2003a).

[1] `http://www.cordis.lu/ist`
[2] `http://cost.cordis.lu`
[3] `http://www.cnc.fr/priamm/`
[4] `http://www.cnc.fr/riam/`

Gómez, Gouyon, Herrera, and Amatriain (2003b, Gómez, Gouyon, Herrera, and Amatriain (2003a) describe content-based music processing tools using MPEG-7.

2.2.2 CUIDADO

The European IST project CUIDADO[5] (Vinet, Herrera, and Pachet 2002a; Vinet, Herrera, and Pachet 2002b; Rousseaux and partners 2002a; Rousseaux and partners 2002b), running from 2001 to 2003, is a follow-up to the earlier CUIDAD project (*Content-based Unified Interfaces and Descriptors for Audio/Music Databases*) running from 1999 to 2001 (the added *O* stands for *online*). It aims at exploiting the MPEG-7 Audio Multimedia Content Description Interface to develop content-based technologies to create the next generation of audio content management systems.

Two pilot applications are targeted which include modules for audio feature extraction, statistical indexing, database management, networking, and constraint based navigation. The first application called *Music Browser* allows to search music by sound similarity, to create musical compilations, to build audio summaries from titles, and to retrieve music according to personal tastes. The second application called *Sound Palette* is an editing tool aimed at professional musicians and sound designers that retrieves audio samples from a large database according to the low-level MPEG-7 descriptors.

The project partners are: Ircam, Sony CSL, Oracle, CreamWare, UPF / Pompeu Fabra University, BenGurion University, Artspages International AS.

2.2.3 ECRINS

The project ECRINS[6] (Mullon, Geslin, and Jacob 2002; Rioux 2001b; Rioux 2001a; Rioux 2002) in the French RIAM network of research support ran from 2001 to 2002. Its focus was on audio analysis in high-level descriptors, permitting efficient retrieval from large on-line databases of sound samples for sound design and multimedia applications, with intelligent automatic classification based on perceptive criteria.

The analysis and representation part of ECRINS is described in (Rodet and Tisserand 2001; Rodet and Tisserand 2002). The project partners are: Digigram, Ina–GRM, Ircam.

2.2.4 Synthesis from High-Level Descriptors

The project *Synthesis and Transformation from High-Level Descriptors* (Lambert 2001a; Lambert 2001b) was carried out by Ircam in collaboration with the CNET research institute of *France Télécom* from 2000 to 2001. It is embedded in the context of the development of an MPEG-4 structured audio authoring tool. The aim is to keep the richness of the original sound for sound resynthesis and transformation, while being able to specify high-level manipulations.

While being more aimed at sound transformation than selection, the results of this project were useful for the understanding of low and high level descriptors.

2.2.5 WEDELMUSIC

The *WEDELMUSIC — Web delivering of Music* project[7] (Wedelmusic 2003) is aimed at publishers and consumers of musical scores, developing an XML-based interchange score format and a full set of tools for building, converting, storing, distributing music on the internet.

[5] `http://www.cuidado.mu`
[6] `http://www.ircam.fr/produits/technologies/ECRINS.html`
[7] `http://www.wedelmusic.org`

2.2.6 MUSICNETWORK

MUSICNETWORK[8], is an IST EC Project, started August 2002. Its aim is to connect the various players in music and multi-media, to coordinate working groups, workshops and conferences on music coding standards and online publishing of music archives.

[8] www.interacactivemusicnetwork.org

Chapter 3

Speech Synthesis

Data-driven concatenative sound synthesis based on unit selection owes much to the last 15 years of research in speech synthesis. Many of the issues are common to both sound and speech synthesis, therefore we describe here the context and different types and algorithms of speech synthesis, and give a brief account of their historical evolution.

Concatenative synthesis techniques were first developed as a part of *text-to-speech* (TTS) synthesis systems. A TTS system transforms written text into a speech signal. It is concerned with all stages of linguistic analysis. The general structure of a TTS system is given in figure 3.1, and a detailed view of the synthesis-method independent linguistic analysis part in figure 3.2. See Allen (1992), van Heuven and Pols (1993), and Pfister and Traber (1994) for an overview and Hess (1996), Holmes (1995), Pols (1992), and Dutoit (1994) for examples and evaluations of existing systems.

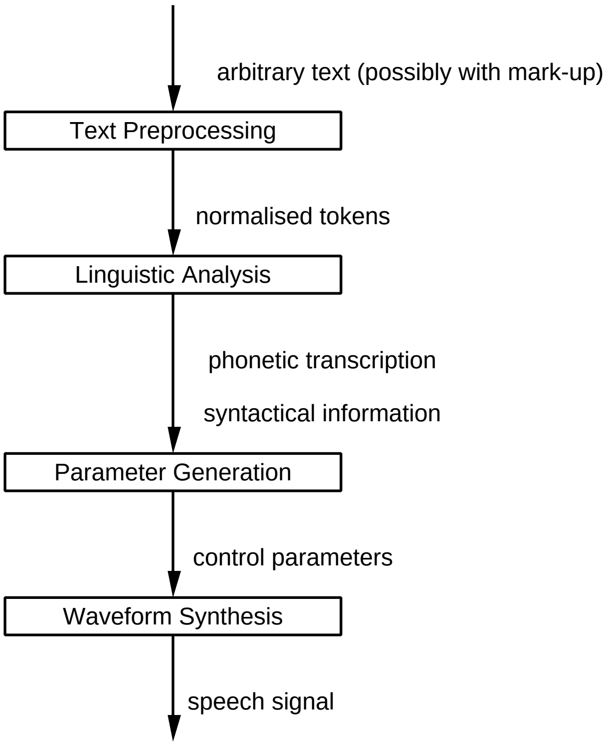


Figure 3.1: General structure of a text-to-speech synthesis system

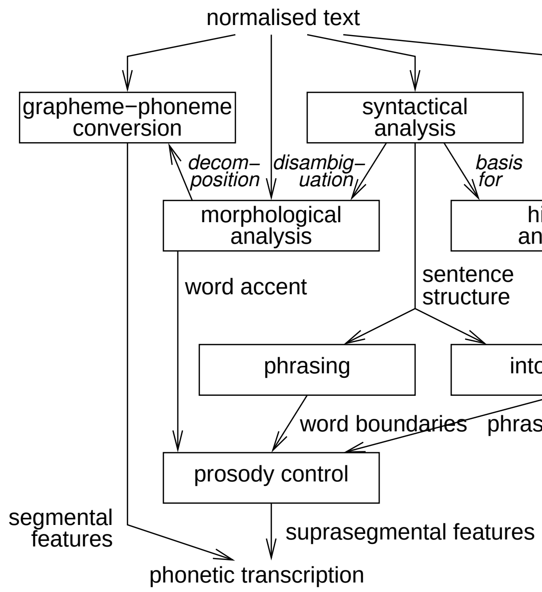


Figure 3.2: Linguistic analysis in a text-to-speech synthesis system

After preprocessing, which essentially performs the tokenisation of the input text stream into words, and replaces numbers and signs by words, the linguistic analysis part performs at least a phonetic, phonological, morphological and syntactical analysis to reconstruct the syntax tree of the phrase tagged with the part-of-speech (POS) information, i.e. the grammatical class, inflection, and function of each word. Linguistic analysis is made much easier when the input text contains already some linguistic or conceptual information. This is sometimes referred to as *concept-to-speech* synthesis. To standardise the text representation of this information, mark-up languages are used, e.g. (Sproat, Taylor, Tanenblatt, and Isard 1997).

The parameter generation part receives this syntactical information, and the string of phonemes making up the phrase, as input to be synthesised. The phonemes determine the desired *segmental features* of each phone. From the syntactical information, the *suprasegmental features* of the phrase are computed, such as the global pitch curve, the pitch accents, the intensity curve, which is called *prosody*. These features are the input for the waveform synthesis part, which outputs the speech signal synthesised according to these control parameters.

For a general account of the state of the art in speech synthesis, interested readers are referred to van Santen, Sproat, Olive, and Hirschberg (1996), for a general introduction to phonetics and phonology, to Clark and Yallop (1996), and for linguistics to Bußmann (1990).

3.1 Classification of Speech Synthesis Methods

3.1.1 Waveform Synthesis Classes

Concatenative synthesis is one of three classes of methods of speech synthesis, shown in figure 3.3. Table 3.1 compares the three classes of methods according to the way the signal is calculated and the control parameters used.

Source–filter synthesis or *formant synthesis* is based on a signal model, where the speech signal is generated by a network of signal processing components such as oscillators, amplifiers, and filters. These simulate the glottal pulse excitation and the resonances of the speech formants (Holmes 1983). *Articulatory synthesis* is based on a physical model where the speech signal is calculated by simulating the acoustic effects of the speech organs, i.e. the acoustic tube formed by the oral and nasal cavities, on a glottal pulse signal. These two classes are often commonly referred to as *parametric synthesis*.

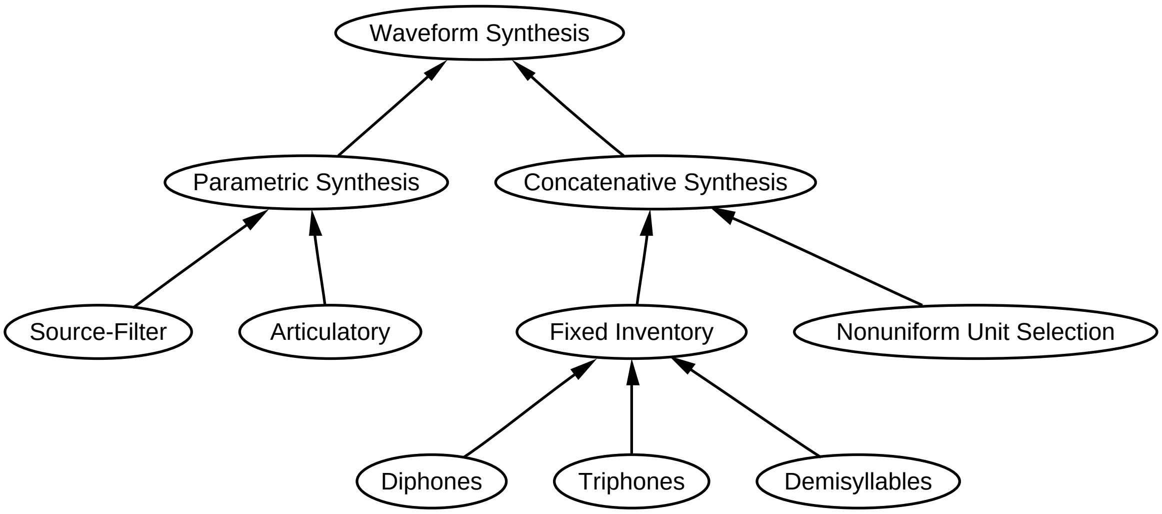


Figure 3.3: Classes of waveform synthesis methods

Synthesis class	Signal calculation	Control parameters
source–filter	signal processing	excitation* and formant parameters
articulatory	physical modeling	excitation parameters, shape of vocal tract
concatenative	unit selection, transformation and concatenation	phoneme type and context, prosodic parameters

Table 3.1: Classes of waveform synthesis methods and their properties

The class of concatenative speech synthesis is further subdivided according to the type and number of synthesis units used into *fixed inventory* and *nonuniform unit selection* synthesis as explained in the following.

3.1.1.1 Fixed Inventory Synthesis

Fixed inventory synthesis uses small units like diphones, triphones, or demisyllables. Diphones (first proposed in (Küpfmüller and Warns 1954)) extend from the middle of a phoneme to the middle of the next phoneme, based on the assumption that the sound is most stable there, and therefore the most easy to concatenate. Demisyllables (first proposed in (Peterson and Sievertsen 1960)) are based on the assumption that concatenation is easiest at syllable boundaries.

The fixed inventory concatenative synthesis methods often use a small, carefully chosen unit inventory, which was necessary for the limited computer memory of the past. Often, there was only one example of each unit (*monorepresented inventory*). However, it has been shown, e.g. by (Portele, Hoefer, and Hess 1996) that this is not enough to adequately model many types of coarticulation effects occurring in many languages. Fixed inventories can be extended with different realisations of the same unit depending on context, which would still have to be carefully chosen, and could never encompass all the prosodic variations occurring in speech. These effects still have to be generated by pitch and loudness transformations of the units, which degrades the quality.

3.1.1.2 Nonuniform Unit Selection

The combined necessities to have many versions of the same phonetic unit (*multirepresented units*) to reduce transformations, and longer units to reduce degradations by concatenation, led to the development of nonuniform unit selection from large speech databases. This was made possible by the growing memory capacity of computers and advances in automatic segmentation and labeling of read speech.

Nonuniform unit selection uses a large database of continuously read speech. The aim of the unit selection algorithm is to select a required unit in the right phonetic and prosodic context, and to select the longest possible unit. The selected units can be single phonemes, diphones, whole words up to parts of phrases. Often, these two aims are contradictory, so that the right balance between them has to be found by adapting weight parameters, to optimise the global quality of the synthesised speech.

3.1.1.3 Rule-based vs. Data-driven Waveform Synthesis

The two parametric synthesis classes of source–filter and articulatory synthesis use the rule-based approach, which presents severe disadvantages: The rules can be very complex, and often hundreds of them are necessary to generate speech that is sufficiently natural. Only short utterances can be feasibly synthesised. If a different voice is needed, a large part of the rules must be changed.

We can see the fixed inventory concatenative synthesis class as partly data-driven: The waveforms themselves are recorded speech data (i.e., changing the voice is easy, only a new recording has to be made), but the prosodic control parameters are still generated by rule. Only the nonuniform unit selection synthesis from large databases is fully data-driven.

3.1.2 Unit Coding

For the representation of the speech units, different encodings can be used. Their choice influences the ease with which transformations can be applied, and the resulting quality of the concatenation.

3.1.2.1 PCM

The simplest encoding uses the sampled speech signal directly. This is sometimes called PCM for *Pulse Code Modulation*. Although retaining the full signal quality, this coding consumes a lot of memory and does not allow for easy modification of the units.

3.1.2.2 LPC and Cepstrum

Mainly of historical interest, *Linear Predictive Coding* (Markel and Gray 1980; Oppenheim 1978) is an early method of efficient *auto-regressive* digital signal processing (Oppenheim and Schafer 1975), developed originally for speech transmission and compression. In parametric source–filter speech synthesis, it can be used to represent the filter part, i.e. the spectral envelope of the voice, that serves to distinguish the phonemes, because it essentially builds up the transfer function of an all-pole filter with a given number of poles (Schwarz 1998; Schwarz and Rodet 1999). The source part, i.e. the excitation signal for the filter, is most often generated parametrically either as a pulse train, or with some glottal waveform model.

When space is an issue, LPC can be used to efficiently compress a unit inventory (see Macon, Cronk, Wouters, and Kain 1997). Serving the same aim of very compact speech coding, *cepstral coding* (Oppenheim and Schafer 1975) of the spectral envelope is used in the system described by Yoshimura, Tokuda, Masuko, Kobayashi, and Kitamura (1999) using a mel-scaled cepstrum analysis (Fukada, Tokuda, Kobayashi, and Imai 1992).

3.1.2.3 HNM

The *Harmonics plus Noise Model* (Stylianou 1998a; Stylianou, Dutoit, and Schroeter 1997; Stylianou 1996; Stylianou, Laroche, and Moulines 1995; Macon and Clements 1995; Macon and Clements 1996; Macon 1996) is the well-known additive harmonic sinusoidal partials plus residual noise parametric signal representation from musical synthesis (see section 10.7 and Risset and Mathews 1969; Serra and Smith 1990; Rodet 1997a) — for once an example where research in musical sound analysis and synthesis gave impulses to speech research, and not the other way round.

The signal is generated by overlap–add (OLA) of the resynthesised and windowed frames, which allows easy and independent modification of pitch, duration, and spectrum.

3.1.2.4 PSOLA

Pitch Synchronous Overlap Add (Valbret, Moulines, and Tubach 1992; Peeters 1998) uses pitch-period-sized windows of the sampled signal with pitch marks, ideally placed on the glottal pulses. The method allows nearly transparent pitch changes by resynthesizing the windows spaced further apart, or closer with more overlap, leaving the basic speech waveform and thus the formants intact. Duration can be independently changed by dropping or repeating windows. PSOLA is a very efficient synthesis method because it works entirely in the time domain.

3.1.2.5 MBROLA and TPMBROLA

Multiband Resynthesis Overlap Add (Dutoit 1993; Dutoit, Pagel, Pierret, Bataille, and der Vrecken 1996; Dutoit 2003) avoids the possible problem of discontinuities in PSOLA by resynthesising the speech units in the database with constant pitch and coherent phases. However, this slightly degrades the sound quality.

TPMBROLA, for *True Period Multiband Resynthesis Overlap Add*, (Bozkurt, Dutoit, Prudon, D'Alessandro, and Pagel 2002) remedies this by resynthesising the database only to the nearest integer pitch.

3.2 Non-Uniform Unit Selection Synthesis Systems

Historically, the need to include the effects of phonetic context into concatenative synthesis lead to the use of multirepresented synthesis units. However, these were still hand-chosen and even multiple versions of the same unit can not cover all sorts of intonation and articulation effects. At the same time, the advantages to use longer units lead to the selection based synthesis systems presented in this section.

Nonuniform unit selection synthesis from large databases was first proposed in (Sagisaka 1988). Many research and commercial systems have been developed since then. A historical overview, based on the compilation in (Prudon 2003), but certainly incomplete, can be found in figure 3.4.

The predecessors for japanese were NTT's system, ATR's *ν-Talk*. CHATR (Black and Taylor 1994) was the first system for english and introduced the classic path-search based unit selection algorithm detailed in section 3.3. All following systems were inspired by CHATR: AT&T's *NextGen TTS* System (Beutnagel, Conkie, Schroeter, Stylianou, and Syrdal 1999), British Telecom's *BT Laureate*, and the work described in (Le Meur 1996) and (Macon, Cronk, and Wouters 1998).

The open source system FESTIVAL (Black and Taylor 1997a; Black, Taylor, and Caley 1998; Taylor 1999) added a focus on flexibility and good software engineering to make research and experimentation easier. This effort is continued in the EULER[1] system (Dutoit, Malfrère, Pagel, Mertens, Ruelle, and Gilman 1998; Bagein, Dutoit, Tounsi, Malfrère, Ruelle, and Wynsberghe 2001; Dutoit 2000) by combining the basis of FESTIVAL with the transformation capabilities and voices of the MBROLA[2] project (Dutoit 1993; Dutoit, Pagel, Pierret, Bataille, and der Vrecken 1996; Dutoit 2003). The most recent developments are the research systems at LIMSI (Prudon and d'Alessandro 2001; Prudon 2003) and at *France Telecom Research and Development* (FTRD) (Blouin, Rosec, Bagshaw, et al. 2002; Blouin and Bagshaw 2003; Blouin 2003), the *BrightSpeech* commercial TTS system by

[1] `http://www.tcts.fpms.ac.be/synthesis/euler`
[2] `http://tcts.fpms.ac.be/synthesis`

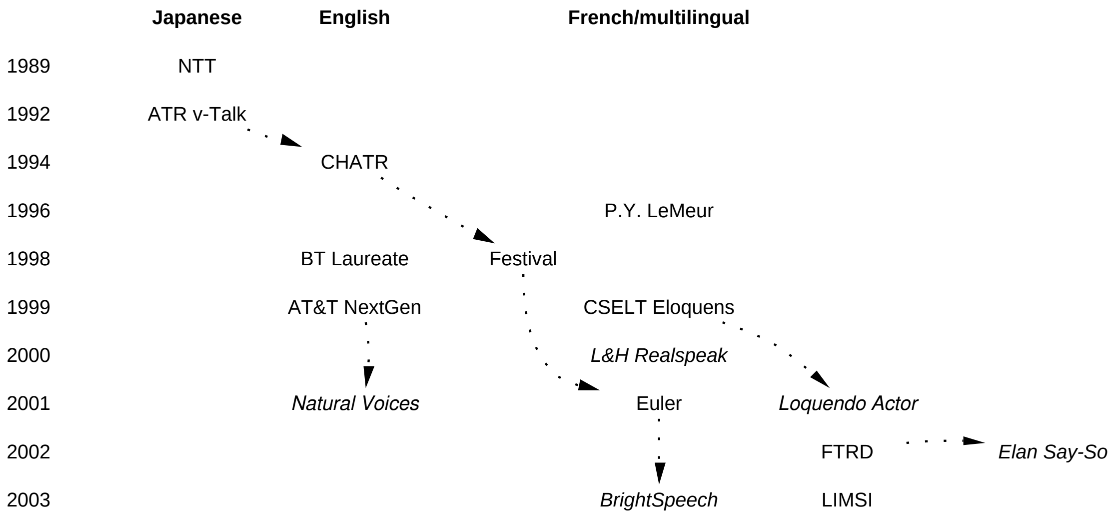


Figure 3.4: Partial historical evolution of nonuniform unit selection speech synthesis systems. Commercial systems are written in italic.

Babel Technologies[3], and the recent commercial systems by *Rhethorical*[4], *Speechworks*[5], *Cepstral*[6], *Nuance*[7], and others.

This explosion of the number of commercial systems is a strong indication of the maturity of the data-driven concatenative speech synthesis technology.

The early systems used the syllable or the phone as basic unit. Because of the easier concatenation, most systems use diphones, however, recent works, e.g. (Blouin, Rosec, Bagshaw, et al. 2002; Blouin 2003) have taken up semiphones as the most flexible unit, from which either phones or diphones can be recombined.

In the interesting recent approach by Tokuda, Zen, and Black (2002), data-driven unit selection is combined with stochastic HMM based parameter generation for source–filter waveform synthesis using cepstral coding for a very compact unit database (Yoshimura, Tokuda, Masuko, Kobayashi, and Kitamura 1999).

3.2.1 Limitations

Although great progess has been made in the last years, some limitations of the state of the art of unit-selection speech synthesis remain: The speech quality and naturalness is very good, but expressivity is still missing. You get what is in the database, and for the databases to be appropriate for synthesis they have to be somewhat uniform. As an example, in the french system described in (Prudon 2003), the database consists of newspaper articles read by professional speakers, so the resulting synthesised speech always has a certain "newsreader" quality. For speech synthesis to deliver emotional content and expressivity as well, the databases have to be much larger than what they are today, and the selection algorithm has to handle more and different features.

First steps towards this direction are described by Bulut, Narayanan, and Syrdal (2002), where four different emotions (anger, happiness, sadness and neutral) were synthesised from specially recorded corpora.

[3] `http://www.babeltech.com/thistory.htm`
[4] `http://www.rhetorical.com`
[5] `http://www.speechworks.com`
[6] `http://www.cepstral.com`
[7] `http://www.nuance.com`

3.3 The Unit Selection Algorithm for Speech Synthesis

In the following, we describe the details of the path search unit selection algorithm first proposed in (Hunt and Black 1996) and used in the speech synthesis system CHATR and the systems influenced by it.

The starting point of the unit selection algorithm is a database of N units u_i and a sequence of T target units t_τ. The unit selection algorithm finds the units from the database that best match the given synthesis target units. The quality of the match is determined by two distance functions, expressed as costs:

The *target cost* C^tcorresponds to the perceptual similarity of the database unit u_i to the target unit t_τ. It is given as a sum of p weighted individual feature distance functions C_k^t as:

$$C^t(u_i, t_\tau) = \sum_{k=1}^{p} w_k^t \, C_k^t(u_i, t_\tau) \tag{3.1}$$

The *concatenation cost* C^cpredicts the discontinuity introduced by concatenation of the unit u_i from the database with a preceding candidate unit u_{i-1}. It is given by a weighted sum of q feature concatenation cost functions C_k^c:

$$C^c(u_{i-1}, u_i) = \sum_{k=1}^{q} w_k^c \, C_k^c \, (u_{i-1}, u_i) \tag{3.2}$$

Consecutive units in the database have a concatenation cost of zero. Thus, if a whole phrase matching the target is present in the database, it will be selected in its entirety, leading to nonuniform unit selection.

The unit database can be seen as a fully connected state transition network through which the unit selection algorithm has to find the least costly path that constitutes the target. Using the weighted target cost $w^t C^t$ as the *state occupancy cost* b_i, and the weighted concatenation cost $w^c C^c$ as the *transition cost* a_{ij}, the optimal path can be efficiently found by a Viterbi algorithm (Viterbi 1967; Forney 1973).

The phonetic class structure of the unit database is exploited by examining only the N_{c_τ} units in class c_τ, indexed by $c_\tau(j)$. However, as is often the case, there can be target units outside of the coverage of the database of a class. In this case, a replacement has to be found, either in a close phonetic class, or by using smaller units, e.g. using semiphones instead of diphones.

for $1 \le j \le N_{c_1}$:
  $a_{j1} = C^t(t_1, u_{c_1(j)})$
for $2 \le \tau \le T$:
  for $1 \le j \le N_{c_\tau}$:
    $a_{j\tau} = \min\limits_{1 \le k \le N_{c_{\tau-1}}} \left(a_{k,\tau-1} + w^c C^c \left(u_{c_{\tau-1}(k)}, u_{c_\tau(j)}\right) + w^t C^t \left(t_\tau, u_{c_\tau(j)}\right)\right)$
    $\psi_{j\tau} = k_{\min}$

The decoding of the path ψ to find the optimal sequence of units s_τ is done in reverse order by first finding the path endpoint k with the least global cost a_{kT}, and then following the backward indices:

$k = \operatorname*{argmin}\limits_{1 \le j \le N_{c_T}} (a_{jT})$
for $T \ge \tau \ge 1$:
  $s_\tau = u_{c_\tau(k)}$
  $k = \psi_{\tau k}$

3.3.1 Optimisations of Unit Selection

The asymptotic computational complexity of the unoptimised algorithm is $O(TN)$ calculations of the target cost C^t, and $O(TN^2)$ calculations of the concatenation cost C^c. With the optimisations described in (Black and Campbell 1995) of using a beam search, complexity is reduced to $O(TNW)$ where W is the beam width (usually 20).

Most of the existing algorithms have in common a training or "learning" phase in which parameters are trained to select appropriate waveform segments for a given set of features. See (Macon, Cronk, and Wouters 1998; Hunt and Black 1996) for an overview, and (Yi 2003) and (Yi and Glass 2002) for efficient weight learning, based on information theoretic criteria. One approach to this step is to partition available data into clusters that can be indexed by the features of the target. This method relies critically on two important principles: discrimination of fine phonetic details using a perceptually-motivated distance measure in training (Ghitza and Sondhi 1997; Hansen and Chappell 1998; Wouters and Macon 1998), and generalization to unseen cases in selection.

An overview of different classes of unit selection algorithms and their optimisations is presented in (Macon, Cronk, and Wouters 1998), notably the automatic clustering of units into a tree (Black and Taylor 1997b; Hunt and Black 1996; Black and Campbell 1995), first used in the Festival TTS system (Black, Taylor, and Caley 1998), and subsequently the interpretation of unit selection as a shortest path search. Optimization of tree-based unit selection is researched in (Black and Campbell 1995; Hunt and Black 1996; Black and Taylor 1997b; Cronk and Macon 1998; Wouters and Macon 1998; Beutnagel, Mohri, and Riley 1999; Black and Lenzo 2001; Blouin and Bagshaw 2003). These articles are concerned with database pruning, search path pruning, automatic determination of the parameters (the weights of the intervening distance functions), and the like. General works on clustering and classification and regression trees are (Breiman, Friedman, Olshen, and Stone 1984; Black and Taylor 1997b; Wang, Campbell, Iwahashi, and Sagisaka 1993; Nakajima 1994).

3.3.2 Optimisations of Concatenation

These is a large amount of work on concatenation for speech synthesis. See the overview in (Prudon 2003). For example Mbrola, PSOLA and Harmonic Plus Noise Model (HNM) (Stylianou 1996; Stylianou 1998a), a comparison of TD-PSOLA and HNM for concatenative speech synthesis can be found in (Syrdal, Stylianou, Garrison, Conkie, and Schroeter 1998). The refinement of concatenaton is treated in the following works: Stylianou (1998b) discusses methods to avoid phase mismatches in HNM based synthesis (section 3.1.2.3), that are the cause of the *buzziness* artefact in synthesised speech. Prudon (2003) finds the best concatenation point by intercorrelation on the borders of the units to concatenate. Compare also the method by Dorran and Lawlor (2003) which find the optimal shift for overlap-add (OLA) for time-scale modification by crosscorrelation.

Chapter 4

Musical Sound Synthesis

Sound synthesis methods fall roughly in the same classes as speech synthesis (section 3.1), as illustrated in figure 4.1: Parametric synthesis and concatenative synthesis are the two large groups. Parametric synthesis could also be called *model based synthesis*, as it is subdivided into synthesis by a signal model and *physical modeling*. The latter is called *articulatory synthesis* for speech. The signal model can be *subtractive synthesis* based on oscillators and filters, as source–filter or formant synthesis is usually called in a musical context, or *additive synthesis*, which is known as Harmonics plus Noise Model (HNM) in speech (see section 3.1.2.3).

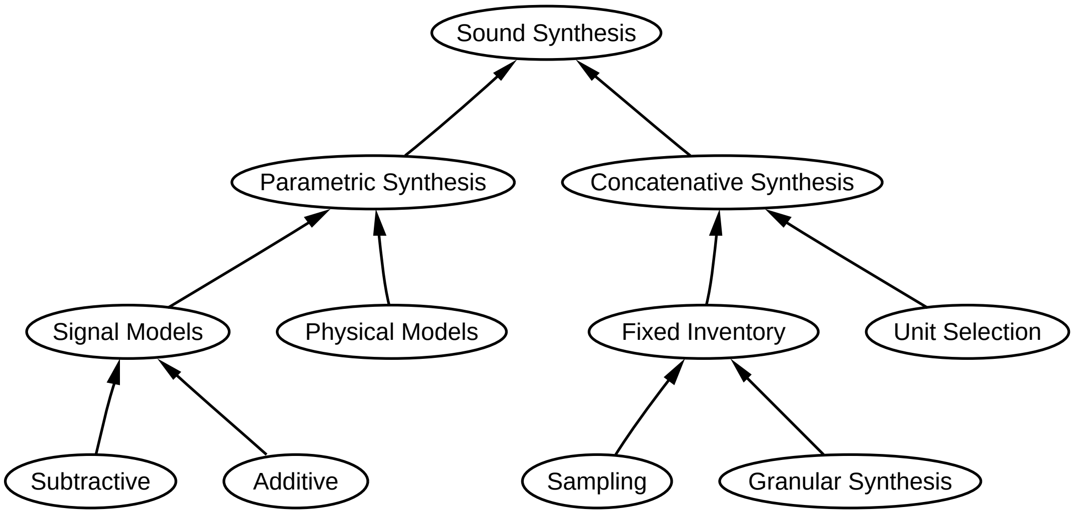


Figure 4.1: Classes of musical synthesis methods

The synthesis of the singing voice occupies an intermediate position between speech and musical synthesis and is briefly presented in section 4.1.

Approaches to musical sound synthesis that are somehow data-driven and concatenative can be found throughout history. They are usually not yet identified as such but the brief discussion in section 4.2 argues that they can be seen as instances of fixed inventory concatenative synthesis.

Since the start of this thesis, numerous other approaches sprang up using ideas of concatenative data-driven synthesis based on unit selection. They are sometimes referred to as *mosaicing* and described in section 4.3. Some of these approaches have found their way into the artistic projects described in section 4.4. I hope to show that all these approaches are very closely related to, or can even be seen as a specialisation of the formalisation of data-driven synthesis proposed in this work, i.e. they could arguably be unified within the Caterpillar framework.

The larger problem of music selection (i.e. selecting a playlist of songs from a music collection) has some related aspects with selection-based sound synthesis, which are detailed in section 4.5. All the approaches of musical sound synthesis are compared in the summary in section 4.6.

4.1 Singing Voice Synthesis

In the synthesis of the singing voice, we find the same classes of algorithms as in speech synthesis (see section 3.1 and figure 3.3): rule-based parametrical synthesis on the one hand, and data-driven concatenative synthesis on the other hand, with subdivisions of the latter according to the number and source of the synthesis units used: a hand-constructed fixed inventory, or unit selection from an automatically constituted database.

See Sundberg (1987) for more detailed information on the singing voice and (Rodet 2002) for an up-to-date overview of current research in singing voice synthesis.

4.1.1 Parametric Synthesis

An overview over parametric synthesis methods is given in Cook (1996), some recent work is described by Henrich (2001).

The Chant project (Rodet, Potard, and Barrière 1984; Rodet, Potard, and Barrière 1985) was originally intended for the analysis and synthesis of the singing voice, but was quickly expanded to cover general sound synthesis by rule. It is based on the flexible and fast time-domain *formant wave forms* additive synthesis method called FOF (from French *Forme d'onde formantique*), introduced by Rodet (1984).

A standard-setting example for artistic singing voice synthesis is the film *Farinelli* (Depalle, García, and Rodet 1994), where the voices of a female coloratura soprano and a male counter tenor were combined to create a new voice with the unique and nowadays unreachable properties of a castrato voice.

The Spasm singing voice synthesis system (Cook 1989; Cook 1991) uses a source–filter model where the parameters for an acoustic tube model of the vocal tract, and for the shape of the excitation signal are adapted from skillful analysis of real singing voice signals. This represents a data-driven approach for the modeling of the signal, but not for the control of the synthesis, which is still rule-based.

Other references for parametric singing voice synthesis systems are Berndtsson (1995), Gershenfeld, Schoner, and Métois (1999), and Hui-Ling (2002).

4.1.2 Concatenative Synthesis

The system developed by Lomax (1996) uses a fixed inventory of diphone singing units, encoded and synthesised by a sinusoidal modeling method (section 10.7).

The Lyricos system (Macon, Jensen-Link, Oliverio, Clements, and George 1997a; Macon, Jensen-Link, Oliverio, Clements, and George 1997b) uses a multi-represented fixed inventory of diphone units. The database consists of ten minutes of specially recorded singing at a low and a high pitch where each diphone occurs in several phonetic contexts to capture coarticulation effects. The units are encoded by a sinusoidal model (section 10.7) and synthesised by overlap-add (OLA). This allows the control of pitch, vibrato amount, vocal effort (rendered by changing the spectral tilt and the breathiness). The selection of the units does not follow the classic Chatr-style path-search algorithm (section 3.3), but proceeds by first selecting the vowel units in the best matching phonetic context, and then filling in the consonants. Synthesis is controlled by a Midi-file (see section 5.4.5) specifying the lyrics in phonetic notation, and all the above expressive control parameters.

The very popular Mbrola project for speech synthesis (see sections 3.1.2 and 3.2, and Dutoit, Pagel, Pierret, Bataille, and der Vrecken 1996) is used as a waveform synthesiser for singing voice synthesis by the BURCAS system (Uneson 2002; Uneson 2003), and abused by the *Melissa* pop music project[1].

Another example of a standard concatenative speech synthesis technique being used for singing voice synthesis is *Flinger*[2], the *Festival Singer* (Macon 2000), using the unit selection of the FESTIVAL speech synthesis system (see section 3.2 and Black and Taylor 1997a) and the LPC synthesiser (see section 3.1.2.2) OGIRESLPC[3] for waveform generation.

Concatenative singing voice synthesis by unit selection is developed by Meron (1999). The selection is performed by a classic CHATR-style algorithm (see section 3.3). The weights of the unit selection algorithm are obtained by an optimised automatic training phase, and a data-driven distance measure, based on phonetic clustering, was used. It is to note that the target for this system was always generated from a real singer's performance, in the spirit of the resynthesis of audio application of concatenative synthesis. This ensures a more naturally sounding synthesis result than from a hand-constructed target, because durations, pitch, accents, articulations and spectral effects are that of a human performance.

The system uses a large database of recordings of one classical singer. To be able to create a database of sufficient size, containing many variants of phonemes, pitchs, and dynamics, commercial recordings of classical pieces for singing voice and piano were used as source material. The material is segmented by alignment of the recordings with their score. Because most classical songs are composed and recorded with accompanying piano, Meron (1999) developed a subsequent separation of the instrument from the singing, based on the score of the accompaniment, and elimination of the instrument by source separation by means of a special model of the piano sound.

Finally, another recent system by Bonada et al. (2001) combines excitation plus resonance modeling with a unit database in sinusoidal coding (section 10.7).

This recent spread of data-driven voice synthesis methods based on unit selection follows their success in speech synthesis, and lets us anticipate a coming leap in quality and naturalness of the singing voice. Regarding the argument rule based vs. data driven singing voice synthesis, Rodet (2002) notes that:

> *Clearly, the units intrinsically contain the influence of an implicit set of rules applied by the singer with all his training, talent and musical skill. The unit selection and concatenation method is thus a way to replace a large and complicated set of rules by implicit rules from the best performers, and it is often called a data-driven concatenative synthesis.*

4.2 Standard Sound Synthesis Techniques Seen as Data-Driven

For musical sound synthesis, We'll start by shedding a little light on some preceding synthesis techniques, starting from the very beginning of electronic music:

4.2.1 Musique Concrète

Going very far back, and extending the term far beyond reason, "data-driven" synthesis started with the invention of the first usable recording devices in the 1940's: the phonograph and, shortly after, the magnetic tape recorder.

At the French national radio, the research group *Groupe de Recherche Musicale* (GRM) of Pierre Schaeffer used for the first time recorded segments of sound to create their pieces of *Musique Concrète*. In the seminal work *Traité des Objets Musicaux* (Schaeffer 1966), Schaeffer defines the notion of *sound object* (*objet sonore*), which is not so far from what is here called *unit*: A sound object is a clearly delimited segment in a source recording, and is the basic unit of composition

[1] `http://www.melissapop.com/`
[2] `http://www.cslu.ogi.edu/tts/flinger`
[3] `http://cslu.cse.ogi.edu/tts/download`

(Schaeffer and Reibel 1967). See Chion (1995) for a guide to the somewhat daunting *Traité des Objets Musicaux*, and Dufour, Thomas, et al. (1999) for a take of contemporary scholars on the work of Pierre Schaeffer and his successors.

4.2.2 Sampling

In the widest reasonable sense of the term, *samplers* (Roads 1996), which appeared at the beginning of the 1980's, were the first "data-driven" sound synthesis devices.

A sampler is a device that can digitally record sounds and play them back, applying transposition, volume changes, and filters. Usually the recorded sound would be a note from an acoustic instrument, that is then mapped to the sampler's keyboard. As soon as memory permitted, *multisampling* was used, i.e. sampling several notes of different pitches, and also played with different dynamics, to better capture the timbral variations of the acoustic instrument than by just changing the playback speed and volume in the sampler.

Modern software samplers, e.g. *Gigasampler*[4], pride themselves to have sampled every note of a piano in every possible dynamic, resulting in 1 GB of sound data. This makes samplers clearly a data-driven fixed-inventory synthesis system, with the sound database analysed by instrument class, playing style, pitch, and dynamics, and the selection being reduced to a fixed mapping of Midi-note and velocity to a sample, without paying attention to the context of the notes played before, i.e. no consideration of concatenation quality.

Similarly, in electronic dance music, a large part of the musical material comes from sampling CDs, containing rhythmic loops and short bass or melodic phrases. As the published CDs number in the tens of thousands, each containing hundreds of samples, a large part of the work consists in listening to the CDs and selecting suitable material.

This development shows a natural drive in professional and multi-media sound synthesis devices or software to make use of the advanced mass storage capacities available today. We can foresee this type of applications hitting a natural limit of manageability. Only automatic support or total automation of the composition process will be able to surpass this limit and make the whole wealth of musical material accessible to the composer or musician.

4.2.3 Granular Synthesis

Granular synthesis (Roads 1988; Roads 1996; Lopez, Martì, and Resina 1998) takes short snippets out of a sound file called *grains*, at a regular rate. These grains are played back with a possibly changed pitch, envelope, and volume. The position and length of the snippets are controlled interactively, allowing to scan through the soundfile, in any speed. If the grain length is longer than the grain period (i.e. the lapse of time between two triggered grains), they overlap in playback, which can lead to "clouds" of grains.

Granular synthesis is rudimentarily data-driven, but there is no analysis, the units size is determined arbitrarily, and the selection is limited to choosing the position in one sound file. However, its concept of exploring a sound interactively could be combined with a pre-analysis of the data and thus enriched by a targeted selection and the resulting control over the output sound characteristics, as described in the free synthesis application in section 17.4.

FOG synthesis (granulation of sound using the FOF envelope) (Clarke 1996; Clarke 2000) is the marriage of the FOF fast and flexible time-domain formant generating envelope (see section 4.1.1 and Rodet 1984) with an arbitrary sound file as carrier signal, producing a effective way to modify the timbre in granular synthesis. With (Clarke and Rodet 2003), FOG now also incorporates the PSOLA envelope (Peeters 1998; Peeters and Rodet 1999b; Peeters and Rodet 1999a; Peeters 2001), allowing further shaping of the grains chopped from the signal. However, for even more control of the sonic result, we do need selection of the source signal, i.e. where to pick the grains that satisfy the wanted sound characteristics.

[4] `http://www.nemesysmusic.com`

4.3 Mosaicing

The term *mosaicing* (the *c* to be pronounced like *k*) is inspired by the idea of the popular "photo mosaics". The principle is the same as in the resynthesis of audio application described in section 17.2: A sound file is given as the target and segmented into arbitrary, usually homogeneous snippets. For each snippet, a matching one is selected from a database of source sound files.

4.3.1 Musical Mosaicing

Musical Mosaicing, or *Musaicing* (Zils and Pachet 2001), developed at the Sony Computer Science Laboratory[5], performs a kind of automated remix of songs. It is aimed at a sound database of pop music, selecting pre-analysed snippets of songs and reassembling them.

Its great innovation was to formulate the unit selection as a constraint solving problem (see section 16.4). The set of descriptors used for the selection is: mean pitch (by zero crossing rate), loudness, percussivity, timbre (by spectral distribution). Work on adding more descriptors has picked up again with (Zils and Pachet 2003).

4.3.2 Soundmosaic

Soundmosaic[6] (Hazel 2001) constructs an approximation of one sound out of small pieces of other sounds (called *tiles*). For version 1.0 of Soundmosaic, the selection of the best source tile uses a direct match of the normalised waveform (Manhatten distance). Version 1.1 introduced as distance metric the correlation between normalized tiles (the dot product of the vectors over the product of their magnitudes). Concatenation quality is not yet included in the selection.

> *The soundmosaic algorithm is: Split the target file up into equal-sized segments, or "tiles". For each tile in the target file, find the closest match in the source files, and replace the target tile with the tile from the source files.*
>
> *For the first demo, the target sound was a recording of a chimpanzee screaming, and the source files were a few short recordings from George W. Bush's public speeches. The final product is a concatenation of soundmosaic results for decreasing tile sizes, starting at a few seconds per tile (such that the first sound is a direct clip of GW's speech), and decreasing to one microsecond per tile.*
>
> Hazel (2001)

4.3.3 MoSievius

The MoSievius system (Lazier and Cook 2003) is an encouraging first attempt to apply unit selection to real-time performance-oriented synthesis with direct intuitive control.

The system is based on segments placed in a loop: According to user controlled ranges for some descriptors, a segment is played or not when its descriptor values lie within the ranges. The feature set used contains voicing, energy, spectral flux, spectral centroid, instrument class. This method of content-based retrieval is called *Sound Sieve* and is similar to the *Musescape* system (Tzanetakis 2003) for music selection (see section 4.5).

[5] `http://www.csl.sony.fr`
[6] `http://thalassocracy.org/soundmosaic`

4.3.4 MPEG-7 Audio Mosaics

In the introductory tutorial at the DAFx 2003 conference[7], titled *Sound replacement, beat unmixing and audio mosaics: Content-based audio processing with MPEG-7*, Michael Casey and Adam Lindsay showed what they called "creative abuse" of MPEG-7: audio mosaics based on pop songs, calculated by finding the best matching snippets of one Beatles song, to reconstitute another one. The match was calculated from the MPEG-7 low-level descriptors (see section 2.2.1), but no measure of concatenation quality was included in the selection.

4.3.5 Sound Clustering Synthesis

Kobayashi (2003) resynthesises a target sound from a pre-analysed and pre-clustered sound base using a vector-based direct spectral match function. Resynthesis is done FFT-frame by FFT-frame, conserving the association of one frame cluster and its predecessor cluster, i.e. the current frame to be synthesised will be similar to the current target frame, and the transition from one frame to the next will be similar to one occuring in the sound data, in the same context. This leads to a good approximation of the synthesised sound with the target, and a high consistency in the development of the synthesised sound. Note that this does not only mean a high continuity, since also transitions from a release to an attack frame are captured by the pairwise association of database frame clusters.

4.3.6 Directed Soundtrack Synthesis

Audio and user directed sound synthesis (Cardle, Brooks, and Robinson 2003) is aimed at the production of soundtracks in video by replacing existing soundtracks with sounds from a different audio source in small chunks similar in sound texture. It introduces user-definable constraints in the form of large-scale properties of the sound texture, e.g. preferred audio clips that shall appear at a certain moment. For the unconstrained parts of the synthesis, a Hidden Markov Model based on the statistics of transition probabilities between spectrally similar sound segments is left running freely in generative mode, much similar to the approach of Hoskinson and Pai (2001) described in section 4.4.1.

4.4 Artistic Applications

4.4.1 Soundscapes

The *Soundscapes* project (Hoskinson and Pai 2001) generates endless but never repeating soundscapes from a recording for installations. This means keeping the "texture" of the original sound file, while being able to play it for an arbitrarily long time. The segmentation into synthesis units is performed by a Wavelet analysis for good join points.

This generative approach means that even the synthesis target is generated on the fly, driven by the original structure of the recording.

4.4.2 Plunderphonics

Plunderphonics (Oswald 1999) is John Oswald's artistic project consisting of songs made up from tens of thousands of snippets from a decade of pop songs, selected and assembled by hand. The sound base was labeled with musical genre and tempo, which were the descriptors used to guide the selection.

> *Plundered are over a thousand pop stars from the past 10 years. Rather than crediting each individual artist or group as he did in the original plunderphonic release, Oswald chose instead to reference morphed artists of his own creation (Bonnie Ratt, etc) It starts*

[7] `http://www.elec.qmul.ac.uk/dafx03`

> *with rapmillisylables and progresses through the material according to tempo (which has an interesting relationship with genre). Oswald used several mechanisms to generate the plunderphonemes that make up this encyclopaedic popologue. This is the most formidable of the plunderphonics projects to date.*
>
> Oswald (1993)

4.4.3 La Légende des siècles

"La Légende des siècles" is a theatre piece performed in 2002 at the *Comédie Française*, using real-time transformation on readings of Victor Hugo. One of these effects, developed by Olivier Pasquet, uses a data-driven synthesis method inspired by this thesis: Prerecorded audio is analysed off-line frame-by-frame according to the features energy and pitch. Each FFT frame is then stored in a dictionary and a clustering is performed using the statistics program *R*. During the performance, this dictionary of FFT-frames is used with an inverse FFT and overlap-add to resynthesize sound according to a target specification of pitch and energy. The continuity of the resynthesized frames is assured by a Hidden Markov Model trained on the succession of FFT-frame classes in the recordings.

4.5 Music Selection

On a larger level, data-driven sound synthesis based on unit selection is related to the problem of music selection from a catalogue: Here, the user wants to select a sequence of songs (a compilation or *playlist*) according to his taste and a certain dramaturgy (a logical evolution from one song to the next). The problem is well described in (Pachet, Roy, and Cazaly 2000), and an innovative solution based on constraint satisfaction is proposed, which ultimately inspired the use of constraints for sound synthesis described in section 16.4.

Other music retrieval systems approach the problematic of selection: The MUSESCAPE music browser (Tzanetakis 2003) works with an intuitive and space-saving interface by specifying high-level musical features (tempo, genre, year) on sliders. The system then selects in real time musical excerpts that match the desired features.

4.6 Summary

As a summary, we can order the abovementioned methods for concatenative musical synthesis according to two aspects, which combined indicate the level of "data-drivenness" of a method. These axes are, first, the structuredness of information obtained by analysis of the source sounds, and second, the degree of automation of the selection. Figure 4.2 shows the place of each of the described methods according to these axes. A third aspect is the inclusion of concatenation quality in the selection, which is expressed in italics in the figure.

We observe certain groups that emanate from this diagram:

- Selection by hand with completely subjective manual analysis (Musique Concrète, Plunderphonics)
- Selection by a fixed mapping from a fixed inventory with some analysis in class and pitch (samplers), or selection by hand with given tempo and character analysis (drum loops)
- Arbitrary source, manual browsing, no analysis (granular synthesis)
- Frame spectrum similarity analysis, target match selection (Soundmosaicing) with a partially stochastic selection (Soundclustering)
- Segmental similarity analysis with stochastic (Soundscapes) or targeted selection (Directed Soundtracks)

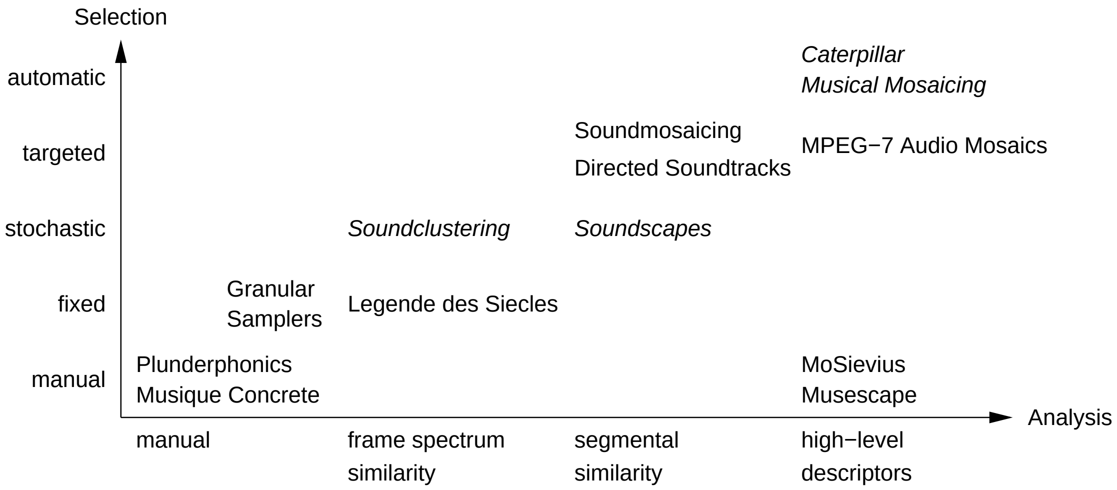


Figure 4.2: Comparison of musical sound synthesis methods according to data-drivenness and use of concatenation (italics)

- Descriptor analysis with manual selection (MoSievius, Musescape)
- Descriptor analysis with fully automatic high-level unit selection and concatenation (Caterpillar, Musical Mosaicing) or without (MPEG-7 audio mosaics)

Part II

Automatic Alignment

Assuming that the music you want to make is a rational function in the z-domain. . .

Yaakov Stein

Chapter 5

Introduction

For selection based synthesis to succeed, we need large databases of musical material, that are well segmented into our synthesis units, i.e. notes in the case of the application to instrument synthesis (see section 17.1). Only automatic methods can provide that much material, but to obtain the required preciseness, blind segmentation methods, such as those described in (Rossignol, Rodet, Soumagne, Colette, and Depalle 1998; Rossignol 2000), are too error-prone. Moreover, with blind segmentation, we'd know the place of the segments, but would know nothing more about their content. That's why much work was devoted to segmentation by alignment, for music where the score was available:

Music alignment is the association of events in a musical score (in our case, notes) with points in the time axis of an audio signal (figure 5.1). The signal is a digital recording of the score being played by musicians and is referred to as the *performance*. An alignment implies a segmentation of the performance according to the events in the score. Additionally, we can associate the symbolic information contained in the score to the segments obtained. Music alignment is sometimes also called *score–performance matching* and has many other applications evoked in section 5.1. A large amount of previous work has been done on music alignment, an overview of which is given in section 5.2.

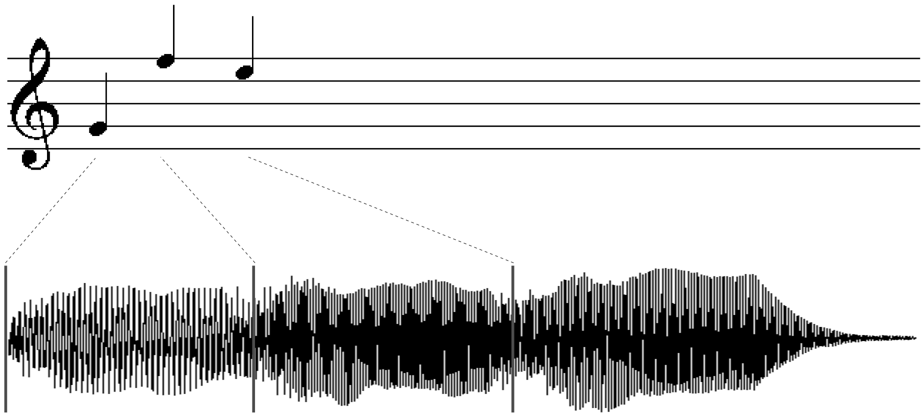


Figure 5.1: The principle of music alignment

The two principal methods used here are Dynamic Time Warping (chapter 6) and Hidden Markov Models (chapter 7). Other methods perform a beat alignment, see for instance Gouyon and Herrera (2003) and Hainsworth and Macleod (2003).

Both methods share a score modeling described in section 5.3, and are based on matching the harmonic partials of the expected notes' pitches with the spectrum of the performance signal. This *Peak Structure Match* (PSM) is explained in section 5.5. This new methodology can cope with polyphonic and multi-instrument performances as well as with performances where fast sequences or trills are present. Normally, blind segmentation methods, which use only the information from the audio signal, are not very accurate with these kinds of performances.

Note that although the sound material to be aligned for inclusion into a unit database is mostly monophonic, almost all instruments can produce polyphonic sounds. Therefore, it is necessary for the alignment algorithm to treat polyphony well, to not get confused in the polyphonic parts of a recording, even though the polyphonic units might never be used for synthesis. Also in view of the other applications mentioned in the following section, our work on polyphonic music alignment is as general as possible.

The alignment is done using a model of the musical score, which is built from an external score coding. To code the score, we have several formats to choose from, which are compared in section 5.4, according to the requirements explained in (Schwarz 2003b). Evaluation of alignment lays the ground for obtaining objective results and their comparison. Evaluation methods are discussed in section 5.6.

5.1 Applications

A great part of the research in computer science is devoted to the automation of processes carried out by humans. For instance, the segmentation of a large collection of recordings, which may last several hours, can not feasibly be done manually because of the large amount of data. The same situation applies for difficult signals (i.e., fast sequences of notes with legato) where manual segmentation may be tedious or imprecise.

Automatic music alignment has many applications in various fields, the most important of these are briefly described here, followed by a comparison of the required accuracy for each of them.

5.1.1 Music Information Retrieval

- Alignment allows the *indexing* of continuous media through the implied segmentation, and labeling of the segments with the score information for content-based retrieval (CBR) (Foote 1999; Dannenberg, Foote, Tzanetakis, and Weare 2001).
- The total alignment cost between pairs of sequences can be considered as a *distance measure* (as in early works on speech recognition), allowing to find the best matching scores or performances from a database.

5.1.2 Musical Analysis

- For a researcher in musicology working on a symbolic score, alignment allows them to listen to a performance of the score at a specific position of interest. See section 2.2.2, Vinet, Herrera, and Pachet (2002b), and Gómez, Gouyon, Herrera, and Amatriain (2003b, Gómez, Gouyon, Herrera, and Amatriain (2003a) for applications.

  Moreover, linking the symbolic score the audio signal allows to test hypotheses about the acoustic effects of compositional and orchestration techniques.
- For the field of *performance study*, the comparison of different performances regarding the expressive parameters related to timing and interpretation characteristics of the musician, can be of great help.

5.1.3 Sound Analysis

- Alignment is related to the problem of real-time synchronisation between performers and computers, usually called *score following*, when additional constraints of low-latency and only local knowledge of the performance are introduced. See section 7.2 for an introduction. Off-line alignment can be used as a bootstrap procedure for the training of real-time statistical models, to get a good segmentation to start from.

- Alignment can help in determining very precise single or multiple *fundamental frequency* values (see section 10.3.4), by reducing the number of candidates according to the expected values from the score. This is applied for example in Rossignol, Desain, and Honing (2001) for precise vibrato estimation. For our application, it comes especially handy for descriptor calculation for unit database building (see the last application below): Many descriptors depend on the fundamental frequency, e.g. the harmonic descriptors described in section 10.7, so a very precise fundamental frequency estimation is essential. This is still not always assured by the current methods, especially in the attack phases and slightly polyphonic parts.

- *Source separation* (Vincent, Rodet, Röbel, Févotte, Gribonval, Benaroya, and Bimbot 2003; Gribonval, Benaroya, Vincent, and Févotte 2003; Vincent, Févotte, and Gribonval 2003) is closely intertwined with segmentation: knowing the temporal location of notes in a multi-instrument piece, for instance, helps singling them out in the frequency domain. On the other hand, to find the notes' time positions, we need to distinguish them from other instruments' notes. A tiny and easily realisable step in this direction is described in section 5.1.6.

5.1.4 Musical Synthesis

- The creation of an *augmented score* (or retranscription) means rewriting the score to be closer to the expressiveness of the performance, e.g. using the aligned note times, the dynamics, and frequency deviations (vibrato) from the performance to write a new Midi file based on the score with e.g. the rhythmic groove and the dynamics of the performance.

- Our main application is the *segmentation* of a performance into notes and labeling (tagging) of the notes with the information from the score for building unit databases. Along with the note pitch and length, there can be additional symbolic information attached to the score, such as dynamics, articulation, or lyrics (see section 10.4).

5.1.5 Required Accuracy

The required accuracy of the alignment varies greatly with the application. Table 5.1 gives a rough estimation of the requirements for the above applications.

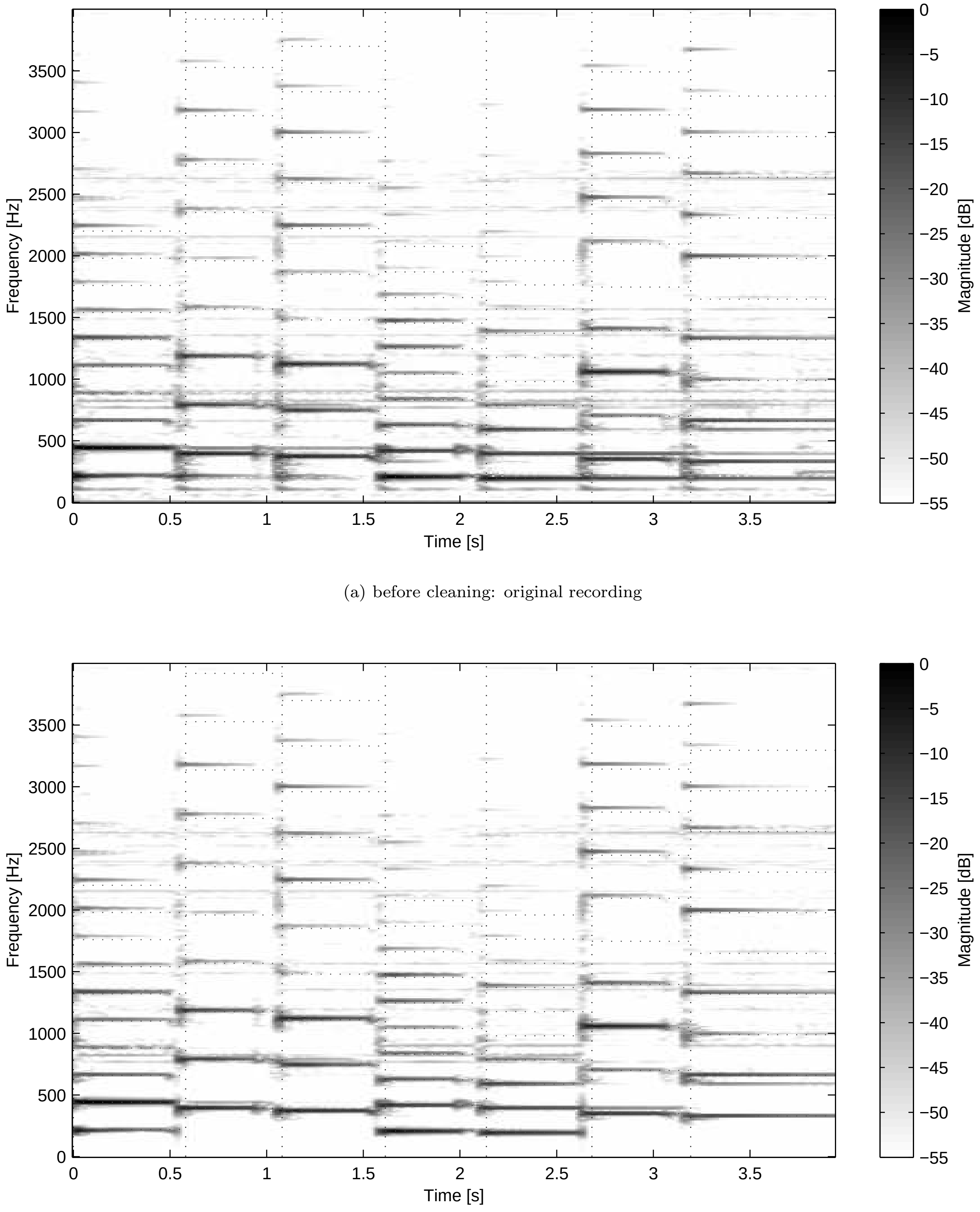
Field	**Application**	**Accuracy**
MIR	Indexing	+
	Distance measure	–
Musical Analysis	Music Browser	o
	Performance study	+
Sound Analysis	Score following	+
	Fundamental frequency estimation	+
	Source separation	+
Musical Synthesis	Augmented score	++
	Segmentation	++

Table 5.1: Comparison of required accuracy of alignment for different applications

5.1.6 Alignment Applied to Source Separation

One possibility immediately following from alignment, which goes a very small step in the direction of source separation, is the demixing or purifying of aligned recordings. One example was realised solely for demonstrative purposes, and not pursued further:

For one example (sound example 6), we split a score-aligned recording of a guitar into its 7 notes, and then filtered each of the notes separately with its harmonic filter bands given by the aligned score (this time filling up the spectrum up to the Nyquist frequency). This leaves only the harmonics



(a) before cleaning: original recording

(b) after cleaning: segmented, filtered note-per-note, and recombined by mixing

Figure 5.2: Cleaning of a guitar recording

proper to each note, eliminating spilled over partials. The filtered notes are then recombined (mixed) to a cleaned recording. The filter band center positions and the spectrogram of the original and filtered signal are shown in figure 5.2. In the original in 5.2(a) we can observe that the partials of the first note and the 5^{th} note at about 2 s continue through to the two next notes because the former were played on an open chord. After filtering, the sustained partials are cleaned out as can be seen in 5.2(b).

For more complex cases (the demixing of multiple instruments), this technique is not refined enough because the filterbands are much too large.

5.2 Previous Work

In general, automatic alignment of sequences has been a popular research topic in many fields, such as string analysis, molecular biology, and notably speech recognition. The literature is considerably vast, and we only mention two comprehensive overviews on the different approaches in speech recognition (Rabiner and Juang 1993) and in biological sequence analysis (Durbin, Eddy, Krogh, and Mitchison 1998).

Alignment has also been used in speech synthesis research, as a useful tool for preparing unit databases for concatenative speech synthesis. The results of the Mbrolign[1] technique (Malfrere and Dutoit 1997b; Malfrere and Dutoit 1997a; Dutoit 1999) from the Mbrola[2] project (Dutoit 1993; Dutoit, Pagel, Pierret, Bataille, and der Vrecken 1996; Dutoit 2003), has been the motivation for our method.

Concerning automatic music alignment specifically, the main works are score following techniques (see section 7.2), mostly based on stochastic models (Grubb and Dannenberg 1997; Grubb and Dannenberg 1998; Raphael 1999b; Raphael 1999a; Raphael 2001a; Raphael 2001c; Loscos, Cano, and Bonada 1999b; Loscos, Cano, and Bonada 1999a; Orio and Déchelle 2001) and specialised off-line alignment methods (Orio and Schwarz 2001; Shalev-Shwartz, Dubnov, Friedman, and Singer 2002; Dannenberg and Hu 2003; Turetsky 2003; Turetsky and Ellis 2003). Most of these techniques consider only monophonic recordings.

For music containing a drum or percussion part, aligning the notes is more difficult for the proposed methods. Beat alignment or note onset detection algorithms (Cemgil, Kappen, Desain, and Honing 2001; Raphael 2001b; Hainsworth and Macleod 2003; Duxbury, Bello, Davies, and Sandler 2003; Gouyon and Herrera 2003) specialise on aligning the drum hits in a performance with a percussion score representation, trying to ignore the interleaved melodic notes.

For general beat analysis and beat tracking methods, see (Scheirer 1998; Lambert 1999; Dixon 2001), although these do not treat music alignment in general.

For note recognition, there are many pitch detection techniques using the signal spectrum (Rodet and Doval 1992; Doval 1994; Prudham 2002) or auto-correlation (de Cheveigné and Kawahara 2002; de Cheveigné and Henrich 2002; de Cheveigné 2002), for instance (see section 10.3.4). These techniques are often efficient in monophonic cases but none of these use score information and are therefore sub-optimal in our situation.

5.3 Score Parsing

In order to build the internal score representation (the model), the external score file has to be parsed. As implicitly introduced in (Orio and Schwarz 2001), the result of the score parsing is a time-ordered sequence of *score events* at every change of polyphony, i.e. at each note start and end, as exemplified in figure 5.3.

Many score files, especially those generated by recording a MIDI-performance, contain score events with slight desynchronisations: For instance, the notes of a chord played on a keyboard are never

[1] `http://tcts.fpms.ac.be/synthesis/mbrolign/mbrolign.html`
[2] `http://tcts.fpms.ac.be/synthesis`

triggered perfectly synchronous, but are slightly arpeggiated (figure 5.4). Equally, passages presumed legato can show a short overlap or gaps between notes (figure 5.5).

To avoid to create too many very short states at these score events, a *quantisation* is performed that fuses score events within a window of usually 30 ms into one single event, and eliminates pauses shorter than 100 ms altogether. The result is that all circled events in figures 5.4 and 5.5 are moved to the time of the earliest event for each circle.

5.4 Score Formats

The scores to be aligned are coded in a digital score file format. The internal score representation and the model is built from this external coding.

5.4.1 Requirements for a Score Format

The definition of the imported score format is essential for the ease of use of automatic alignment. The requirements and constraints are multiple:

1. It has to be powerful, flexible, and extensible enough to represent all the things we want to align. These are essentially the notes, but some exceptions apply: trills, for instance, should not be represented as the notes actually played, but as a single object. For singing voice performances, there are musical events that are insufficiently represented as notes, e.g. fricatives, because they have no pitch.

   Another important information are the *cues*. A cue is a numeric or symbolic label attached to a position or a note in the score, that is output by a score following system to trigger an event in the electronic part of the piece whenever the corresponding score position is reached in the performance.

2. The score format should correspond as closely as possible to the printed score the musician had at her disposal, when she performed the score. It should be, above all, on the same level of abstraction, i.e. it should bear easily accessible high-level musical information intended for the musician.

3. It should be easy to generate. Export from popular score editors, or recording it from live input should be easily possible.

4. It should be easy to read: There should be an existing parser for importing it, preferably as an open source library under a license qualifying for free (libre) software[3].

5. It is convenient to be able to fine-tune imported scores within the alignment system, without re-importing.

6. A large corpus of pieces coded in this format should be available, covering at least classical music where recordings are readily available.

In the following, we give an overview of several types of current score-representation formats, with remarks about their suitability for music alignment, and a conclusion in section 5.4.6. For an overview of all sorts of musical data formats, see (Selfridge-Field 1997).

[3]The free software community (see `http://www.fsf.org`) is about to adopt the term *libre software* to distinguish the sense of “free” as in “free speech” from the “free beer” sense.

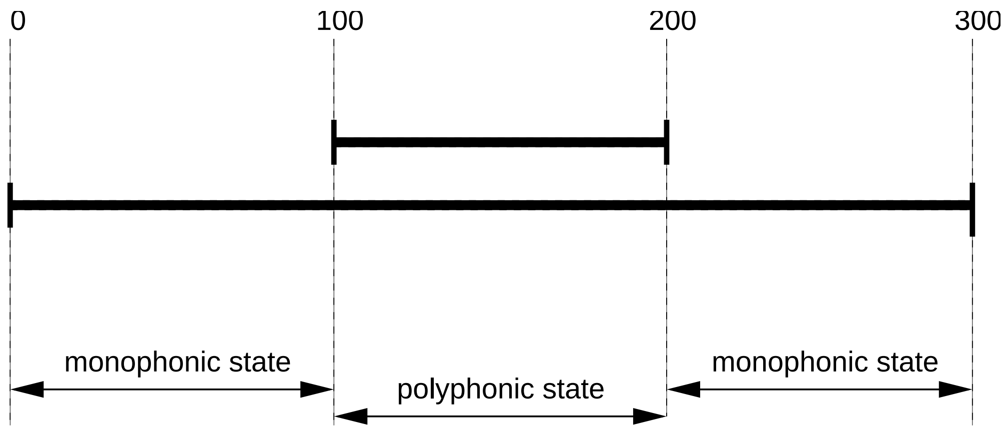


Figure 5.3: Score parsing into score events and the states between them.

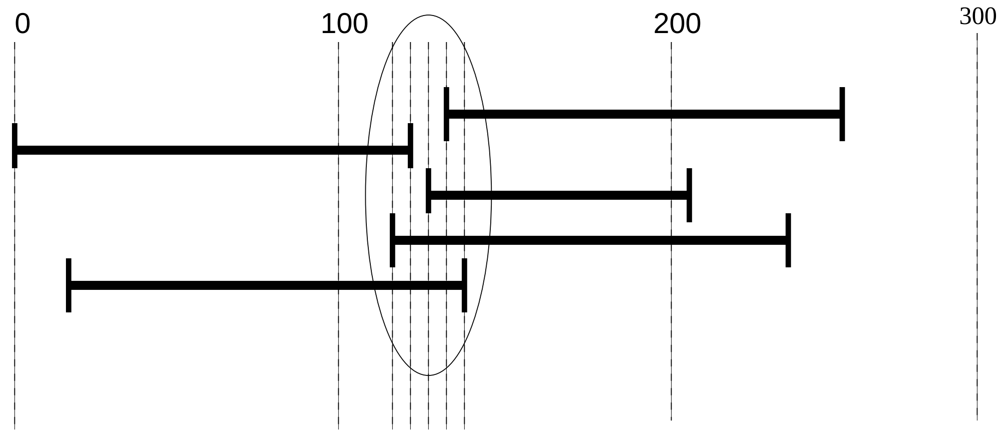


Figure 5.4: Desynchronised chord.

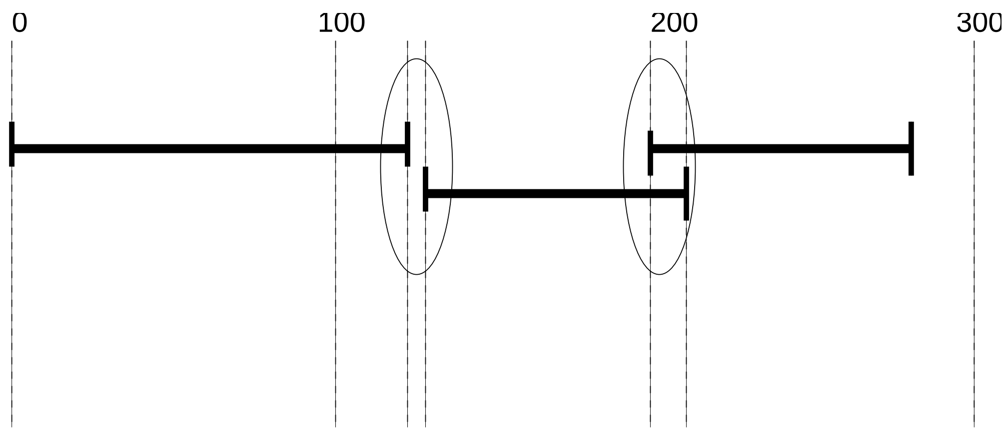


Figure 5.5: Desynchronised legato notes.

5.4.2 Score Editor Formats

Finale, Sibelius

Finale by *Coda Music*[4] and *Sibelius* by *Sibelius*[5] are the most widely used commercial score editors. Their file formats are proprietary and even if they could be decoded, they would most probably not be suitable for high-level score coding, since they focus on graphical printed score layout.

NIFF

Notation Interchange File Format (NIFF Consortium 1995) is a graphical interchange format for music notation, and therefore not suitable either for abstract score representation.

Guido

Guido (GUIDO 2003; Hoos, Hamel, Renz, and Kilian 1998) is an open non-commercial format for high-quality score layout and representation. Its representational capabilities make it applicable for music information retrieval (Hoos, Renz, and Görg 2001). Free software tools and libraries for reading, displaying, and editing exist.

5.4.3 Mark-Up Languages

Mark-Up languages represent musical scores in a specialised format, or document type, of a standardised generic meta-language, such as SGML (Standard Generalized Markup Language) or XML (Extensible Markup Language, Cover 2000; DuCharme 1999). These are ASCII-based tagged file formats with the advantage that the parsing and validation can be handled by freely available open source libraries, and only the music-representation-specific logic has to be taken care of by the application. This leads to a high interchangeability between applications.

Standard Music Description Language (SMDL)

The Standard Music Description Language (Newcomb 1990; Newcomb 1991; ISO 1995) is a ISO standard developed by the Music in Information Processing Standards (MIPS) committee to enable interchange of musical documents, and attach external information to cues for musical comedy and opera. It is an SGML document type definition (DTD), i.e. a format defined in the meta-language SGML and, due to its high complexity, not very widely used in the community.

MuTaTeD

Music Tagging Type Definition (MuTaTeD!)[6] is the integration of SMDL (Standard Music Description Language) and NIFF (Notation Interchange File Format) to support the representation of music as a time-structured entity and at the same time to provide a standard for high-quality display via the NIFF format. MuTaTeD'II[7] (Böhm, MacLellan, and Hall 2000), started in November 1999, adds music information retrieval capabilities with delivery/access services for encoded music. The wider aim is to establish a standard Meta-DTD for music tagging languages.

Wedelmusic XML Format

The Wedelmusic XML Format (Bellini and Nesi 2001) has been developed within the *WEDELMUSIC—Web Delivering of Music* project[8] (Wedelmusic 2003), see section 2.2.5. It offers a standalone editor, and plugins to save Wedelmusic XML format files for *Finale* and *Sibelius*.

MusicXML

MusicXML[9] is an open standard for high-level score representation in the form of an XML DTD (Recordare 2003). Its aim is to be an interchange format for notation, analysis, retrieval,

[4] `http://www.codamusic.com`
[5] `http://www.sibelius.com`
[6] `http://www.music.gla.ac.uk/CMT/projects/MuTaTeD1/index.html`
[7] `http://www.music.gla.ac.uk/CMT/projects/MuTaTeD2/index.html`
[8] `http://www.wedelmusic.org`
[9] `http://www.musicxml.org/xml.html`

and performance applications for common Western musical notation from the 17$^{\text{th}}$century onward. There are already plugins to write MusicXML from Finale, Sibelius, and many other programs. For reading, one of the standard free XML libraries and the free DTD are sufficient. With the growing number of import filters, academic corpora can be converted to MusicXML.

Music Notation in MPEG

In August 2003, the Moving Picture Experts Group (MPEG) has started an Ad Hoc Group (AHG) on music notation requirements[10] to study the requirements of integrating music notation into MPEG. This activitiy is supported by the MUSICNETWORK[11] project (section 2.2.6) in workshops about music notation[12].

A list of the requirements for a score representation or music notation format for the application of music alignment and real-time score following has been submitted to the AHG in (Schwarz 2003b), to be included in the process of defining such a standard.

5.4.4 Frameworks

A framework is a library of classes or functions destined to facilitate writing music applications. Some existing frameworks also provide data-structures for musical score representation, access and manipulation functions for these, and an external file representation, which makes them eligible for the score format we are looking for. However, although they are quite high-level, their principal aim is not interchange, and due to their strong link with an existing application there are doubts about their generality and extensibility.

Common Practice Notation

Common Practice Notation (Maidín 1999) is the file format of the CPN View music editor, but not widely used otherwise.

Allegro

The Allegro music and sound processing framework (Dannenberg and van de Lageweg 2001) defines a high-level internal data structure for musical scores that can be saved to disk and reloaded.

5.4.5 MIDI

The *Standard MIDI File* (SMF) format (MIDI Manufacturers Association 2001) is an extension to the MIDI (*Musical Instrument Digital Interface*) control protocol (MMA 2003) to represent MIDI performances in a file. It adds to the pure real-time MIDI control messages the possibility to group them into several tracks, to represent tempo and musical meter, and to add metadata (comments, copyright, author information) and lyrics.

5.4.6 Conclusion

Unfortunately, MIDI is currently still the most practical format, even with its shortcomings and restrictions. It does indeed fulfill all the requirements mentioned above: Defining some conventions, it can code everything we want to align, e.g. using special Midi channels, controllers, or text events[13]. It can be exported from every score editor, and can be read and fine-tuned in many simple and embedded midi editors.

The result is that we stay with Midi for the time being. There is an overwhelming number of MIDI files available on the net (however, sometimes in bad quality[14]). For all the other formats, although

---

[10] `http://www.dsi.unifi.it/\home{nesi}/mpeg/ahg-mn-65-66.html`

[11] `http://www.interacactivemusicnetwork.org`

[12] `http://www.interactivemusicnetwork.org/events/notation_workshop_2003.html`

[13] We could—if we really wanted to—use text events to carry XML tags, such as to code structured information in a canonical extensible way, linked to precise points in time in the midi score.

[14] This constitutes good test examples for score–performance mismatches, see section 6.5.2ff for such a case.

much more advanced, the lack of sufficiently large corpora is the most severe disadvantage. The latter condition, however, could change rapidly as soon as the musical information retrieval (MIR) and musicology communities decide to use one of the formats, and are able to gain institutional support from several sufficiently large research facilities, and, more important, commercial support from one the leading score editor vendors.

However, there have been very recent encouraging events in the long research for a higher-level representation that inserts itself well into the composer's and musical assistant's workflow: The MPEG consortium has formed an *Ad Hoc Group on Music Notation Requirements* to investigate for the inclusion of music notation into the MPEG-4 multimedia standard, and MusicXML is gaining ground and supports already a large number of applications for input and output.

As MusicXML is already an existing standard, it will be supported by our further development, but our involvement in the MPEG AHG (Schwarz 2003b) will ensure that a possible MPEG standard for music notation—which would have substantial commercial backing—will meet our requirements for a score format for alignment.

5.5 Peak Structure Match

The principal feature for segmentation of musical signals, independent of the method used, is pitch (as opposed to spectral envelope for speech). However, pitch tracking is still error prone, even more so for polyphonic signals. This is why we do not use pitch as a feature directly, but the structure of the peaks in the spectrum given by the harmonic sinusoidal partials. This extends well to polyphonic signals.

The immediate idea how to represent the peak structure is to use the peaks of the harmonic sinusoidal partials directly, but this again relies on a detection of the fundamental frequency and costly spectral peak detection. Orio and Schwarz (2001) proposed therefore the use of bands in the FFT magnitude spectrum.

The expected peaks are modeled as an expected FFT magnitude spectrum S from the pitches in the score: For each note with pitch f running at a certain score frame, h harmonic peaks at frequencies $k \cdot f$, $1 \leq k \leq h$, are generated. After a number of tests, we chose $h = 8$ but good results can be obtained also with smaller values. The peaks take the form of rectangular spectral bands with an equal amplitude of 1 in an otherwise zero spectrum. Figure 5.6 shows a set of bands for the monophonic case, together with a good matching performance spectrum in 5.6(a), and bad matches in 5.6(b) and 5.6(c). Figure 5.6(d) show the easy extension to polyphonic matching with the harmonic bands of two notes. Each band has a bandwidth of one half-tone to accommodate for slight tuning differences and vibrato. This generated score spectrum S is multiplied by the Fourier magnitude spectrum P of one frame of the performance. If the peak structures of the two frames are close, the element-wise product $S \cdot P$ will be high. We can also see this procedure as filtering and the generated spectral bands as frequency-domain filter bands.

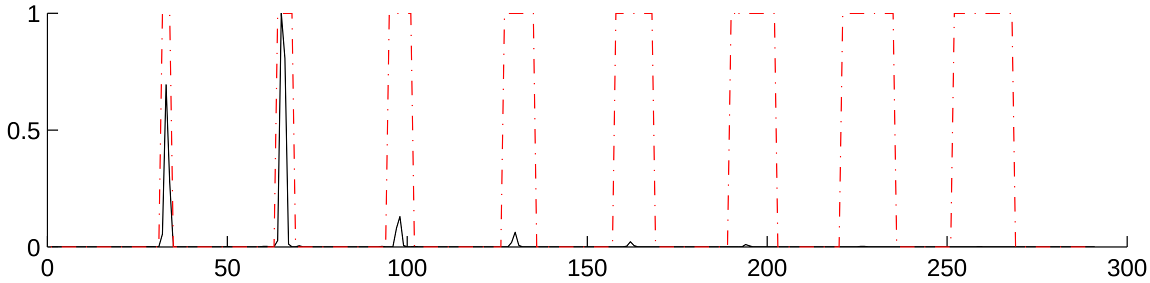
Normalization of the results of $S \cdot P$ is necessary to prevent a loud, noisy frame from matching all generated bands, as depicted in figure 5.6(c). To prevent this, we divide its value by the signal energy in the frequency range that contains all the rectangular spectral bands. For score state n, this range is given by the FFT bin indices between $i_{\text{low}}(n)$ and $i_{\text{high}}(n)$ as

$$R(n) = \{i \in \mathbb{N} | i_{\text{low}}(n) \leq i \leq i_{\text{high}}(n)\} \tag{5.1}$$

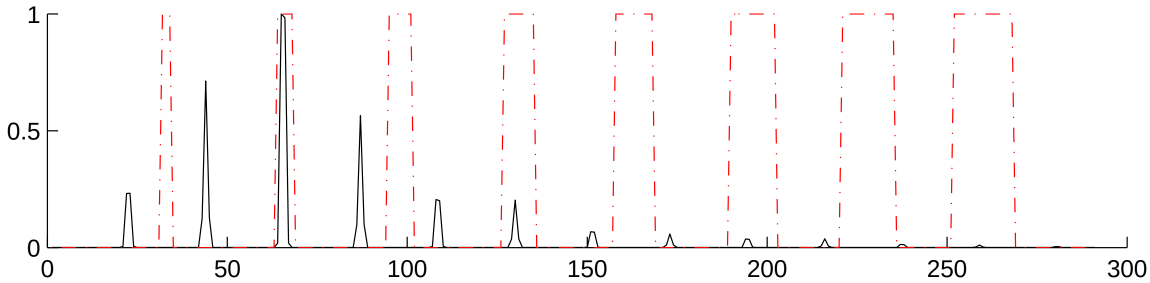
The definition of the peak structure match is thus

$$PSM(m,n) = \frac{\sum\limits_{i \in R(n)} S_i P_i}{\sum\limits_{i \in R(n)} P_i} \tag{5.2}$$

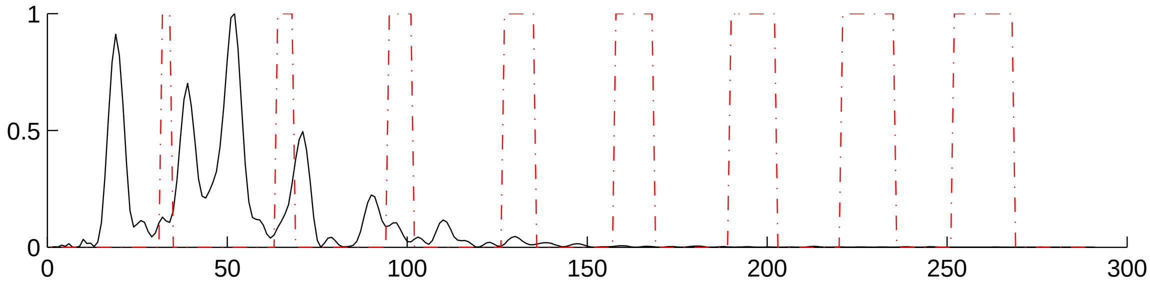
with m, n the frames in the performance and in the score, respectively, and i the FFT bins. The PSM takes values between 0 (when no energy at all lies in the expected bands) and 1 (when all



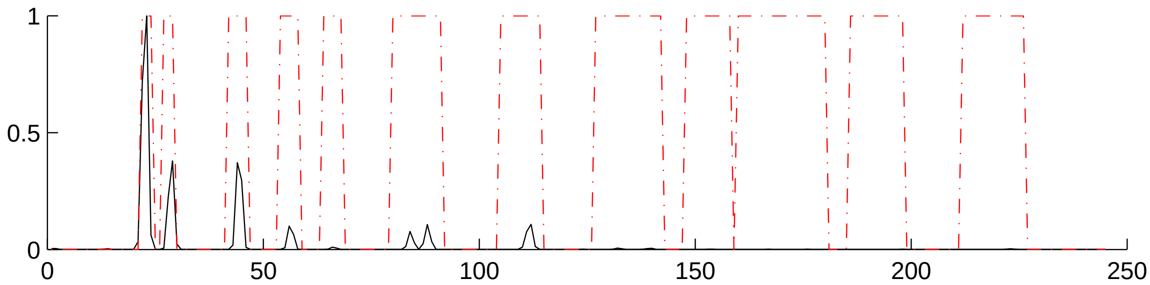
(a) A good matching performance spectrum.



(b) A bad match, most peaks are outside of the harmonic bands.



(c) A bad matching attack frame.



(d) Good match of a chord of two notes.

Figure 5.6: Examples of the generated harmonic filter bands with performance spectra.

energy in $R(n)$ is in our filter bands). For the dynamic time warping algorithm, we simply invert the peak structure match and express it as a *peak structure distance*:

$$PSD(m,n) = 1 - PSM(m,n) \tag{5.3}$$

The calculation of the product $S_i P_i$ for the PSD can be implemented very efficiently by summing the bins of P within the bands, because the bins of S are either 1 within a band, or 0 outside:

$$\sum_{i \in R(n)} S_i P_i = \sum_{i \in R(n) \cap B(n)} P_i \tag{5.4}$$

with

$$B(n) = \{i \in \mathbb{N} | S_i = 1\} \tag{5.5}$$

the selected FFT bins for score frame n.

5.6 Evaluation of Alignment

Evaluation gives an indication of the quality of the alignment algorithm and allows to compare different methods, parameters, etc., and to quantify the improvements gained by training. However, often the best alignment is not clear and dependent of the use made of it. As introduced in (Schwarz and Orio 2002; Orio, Lemouton, Schwarz, and Schnell 2003), we can distinguish between subjective vs. objective evaluation, detailed in sections 5.6.1 and 5.6.2. Some considerations and work on an evaluation framework, i.e. methods and tools how to perform the evaluation are presented in section 5.6.3. The evaluation of score following and alignment systems was the subject of a panel session at the *International Computer Music Conference* (ICMC) 2003. A brief account of the conclusions reached is given in section 5.6.4.

5.6.1 Subjective Evaluation

A *subjective* or *qualitative* evaluation of an alignment system verifies that the important performance events are aligned with a level of errors that is good enough for the use made of it, which is therefore dependent on the requirements of the application. Independent of the application, it can be done by assuming the hardest case, i.e. all score events have to be aligned precisely (whatever that means). There are different ways to perform a human evaluation, some of which may seem trivial, but need reflection and experience nonetheless. For the methods using evaluation by listening, sound examples can be found on the CD, based on an alignment of sound example 2.

- First of all, displaying the alignment marks along with the waveform, as in figure 5.7(a), gives a first overview of the alignment. Note that only in simple cases we can see anything useful on the waveform display, and then again, these cases are usually well aligned. Compare for example with the waveform displays in figures 6.6(a)ff on pages 59ff. Zooming in helps, even more so playing the sound from a position by mouse click (playing from the alignment mark is not enough, since many instruments still sound reasonable even when a bit of the attack has been cut off).

- Displaying the alignment marks and the expected harmonic frequencies overlayed on a spectrogram, as in figure 5.7(b), allows to better recognise the coincidence of the alignment with attacks, which are usually well distinguishable by a human, even for the more complex examples in figures 6.6(b)ff. Note however that, due to the windowing effect, in the spectrogram view the alignment seems in general to be a little bit too late, although we can see in the waveform view in figure 5.7(a) that this is not the case in general. This method goes further than the first one, but in very complex cases, like in the Mozart quartet in figure 6.8(b) on page 59, some weak notes are hardly distinguishable, e.g. the three bass notes only barely visible even in the extreme zoom in figure 6.2(a).

- For our application of unit database building, we can evaluate the quality of the segmentation by listening to the performance with short pauses inserted at every alignment mark (sound example 3). This reveals if the note is cleanly cut, with no overspill of the last, or intrusion of the attack of the next unit.
- A less tiring, but also less precise method is to listen to the performance and a click that is output at each alignment mark in parallel, verifying that the click coincides with a note (sound example 4). This automatically includes the human perceptual thresholds for detection of synchronous events in the evaluation process.
- Last, we can retranscribe the score to use the timing of the alignment, and write the aligned score to a MIDI file (sound example 5). Listening to the MIDI file and the performance in parallel, possibly on separate stereo channels, should give as an overview of the alignment quality. However, we have to deal with too much unnecessary acoustic information, that can hinder the evaluation: The choice of the sounds for the MIDI rendering, the pitches, which can be slightly out of tune between the MIDI instruments and the recorded instruments, vibrato that is missing, etc., make this method of subjective evaluation less effective.

5.6.2 Objective Evaluation

An *objective* or *quantitative* evaluation, i.e. to know down to the millisecond where each performance event was recognised, even if overkill for most actual uses of alignment, is helpful for refinement of the technique and comparison of different alignment algorithms, implementations, and parameterisations, quantitative proof of improvements, automatic testing in batch, making statistics on large corpora of test data, and so on.

Objective evaluation needs reference data that provides the correct alignment of the score with the performance. Providing the reference alignment by hand is tedious. So far, no pre-aligned test databases with musical performances exist. Synchronous recordings of the audio and the MIDI output of midified instruments (flute, piano) are a good way to obtain the performance/reference pairs because of the perfect synchronicity of the data.

For the special case speech, a reference alignment for speech or singing voice can be obtained by full scale speech recognition with a phonemic transcription of the spoken or sung text. A speech aligner such as Mbrolign (Malfrere and Dutoit 1997b; Malfrere and Dutoit 1997a) then aligns the speech recording to a synthesised version of the text to obtain the exact alignment.

The reference alignment positions t_i^r are then compared to the ones found by the alignment algorithm t_i^a for score events $1 \leq i \leq N$. In fact, due to score–performance mismatch and misses or false matches for the real-time case, the events recognised by the alignment algorithm do not necessarily correspond one-to-one to the reference events.

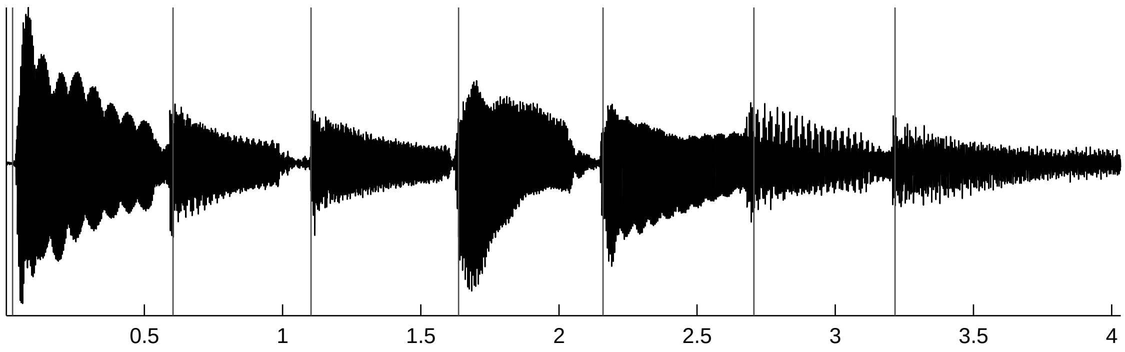
Several measures are taken to quantify the alignment result, based on the *offset* d_i which is defined as the time lapse between the alignment positions of corresponding events:

$$d_i = t_i^r - t_i^a \tag{5.6}$$

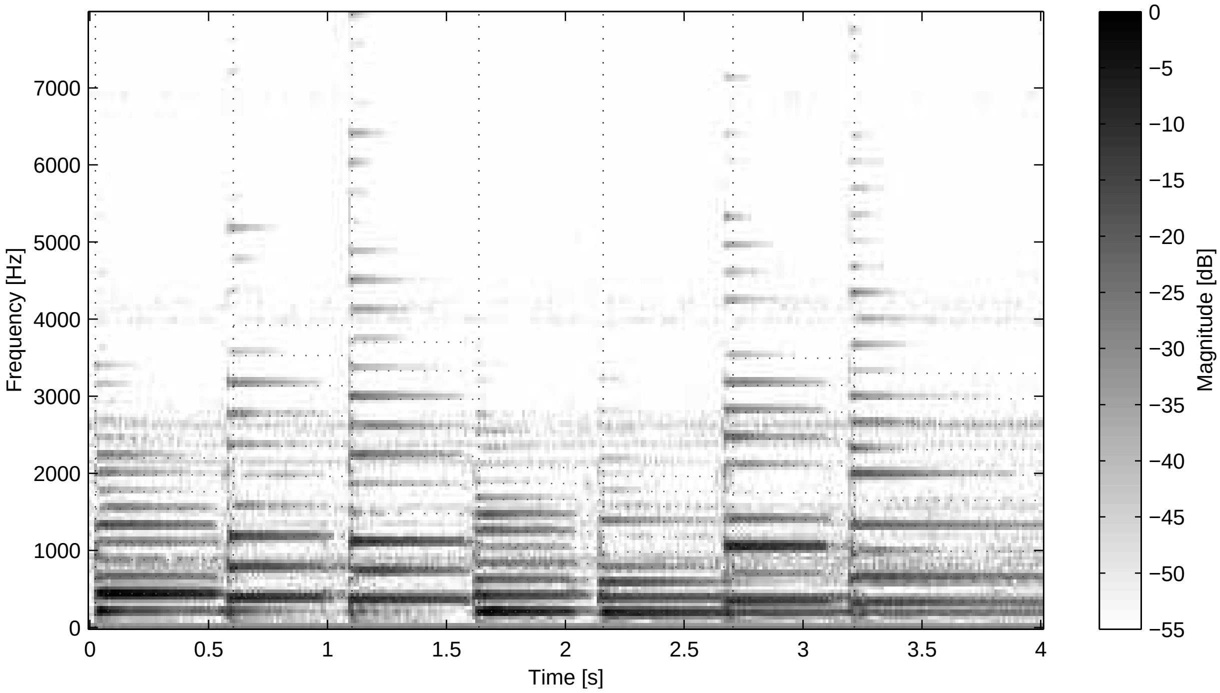
The values characterising the quality of an alignment are then:

Miss rate
: According to classical measures of automatic systems that simulate the human behaviour (Beeferman, Berger, and Lafferty 1999), alignment errors can be due to the *miss* of a correct label at a given moment, or to the *false alarm* of an event incorrectly given. The miss rate p_m is the percentage of missed score events.

False detection rate
: The false detection rate p_f is the percentage of events aligned where no event was expected in the reference.



(a) Waveform and alignment marks



(b) Spectrogram and alignment marks

Figure 5.7: Alignment result example for an easy Guitar melody

Error rate
: Events with their absolute offsets $e_i = |d_i|$ greater than a certain threshold θ_e (e.g. 100 ms), or events that have not been output by the follower (the misses), are considered an error. The error rate p_e is the percentage of these error events.

Average offset
: The average offset for non-error cues μ_d, if different from zero, indicates a systematic latency.

Variance of offset
: The standard deviation of the offset for non-error cues σ_d shows the imprecision or spread of the alignment.

Average error
: The average absolute offset μ_e of non-error cues shows the global precision.

There are other aspects of the quality of an alignment or score following algorithm not expressed by these values: e.g. the number of cues detected more than once, by zig-zagging back to an already detected cue.

Again, the tolerable number of mistakes and error of the aligner largely depend on the kind of application and the type of musical style involved. Note that, for this kind of evaluation, it is assumed that the performance matches the score, i.e. the musician played what was written. It is frequent that, for real applications, mismatched scores or performances will occur, suggesting as another measure the time needed by the aligner to recover from an error situation, that is, to resynchronise itself after a number of "wrong" notes are played (see section 6.5.3 for an example). The tolerated number of mismatching notes, without disturbing the alignment of the matching notes, is another quality measure, that can always be experimentally measured through generation of wrong scores to be aligned with the same performance.

5.6.3 Evaluation Framework

Evaluation is best implemented outside of the alignment system, instead of instrumenting the insides of the aligner by inserting measurement code. The only interface with the alignment system is a result file in a well-defined format, such as SDIF (section 13.4.1), with all information about the score, the alignment and the reference. This *black box testing* approach has the advantages that it is then possible to test and compare other aligners or old versions of the alignment algorithm, and to interchange result data with other research groups.

For the DTW alignment (chapter 6), the result is written to an SDIF file after completion, using the MATLAB–SDIF interface described in section 13.4.3. For the real-time HMM aligner (chapter 7), we need an external *jMax*-object that collects the data and writes it to the result file, using the *jMax* SDIF streaming objects described in section 13.4.3.

However, with the opposite *glass box testing* approach of adding evaluation code to the alignment system, is is possble to inspect its internal state (but which is not necessarily comparable with other alignment algorithms) to debug and optimise the algorithm.

5.6.4 ICMC 2003 Panel Session on Evaluation

The International Computer Music Conference in September 2003 hosted a 90 minute panel session with the topic of evaluation of score following and audio alignment systems in the aim of creating a standard for evaluation and a standard set of test cases. It was organised by Noel Zahler, the participants were Roger Dannenberg, Cort Lippe, Ozgur Izmirli, Xavier Rodet, and Diemo Schwarz. Nicola Orio, Miller Puckette, Christopher Raphael, and Barry Vercoe were also invited but could not attend.

Each representative of a score following project gave a short synopsis of their research and the results of that research to date, followed by the methods used to evaluate the research and suggestions for developing a standard. After the presentations, there was a moderated discussion on the components for creating a mechanism for standardization.

The proposals in sections 5.4 and 5.6.2 brought forth by the author regarding score formats and evaluation measurements were adopted as a starting point and over the following year, the participants will exchange the necessary data to agree on such a standard and constitute a database of test sounds and their reference alignments. The medium for the discussions will be the mailinglist *score-recognition*[15], administered by the author.

[15] `http://list.ircam.fr/wws/info/score-recognition`

Chapter 6

Dynamic Time Warping

One widely used method for automatic alignment is Dynamic Time Warping (DTW). This technique finds the best global alignment of two sequences, based on local distances. It uses a Viterbi path finding algorithm that minimizes the global distances between the sequences. DTW for speech recognition is described in detail in (Rabiner and Juang 1993).

The sequences to be aligned consist of frames containing features. The feature data for the performance are extracted by signal analysis techniques. The feature data for the score are generated for each frame according to a model of the instrument. In our case, the model is a simple harmonic spectrum that is constant for each note, together with a model of the attack and a model for silence.

Alignment by DTW is carried out in the following steps:

1. Construction of the score representation by parsing of the MIDI file into *score events* as defined in section 5.3, and building of the score model.
2. Extraction of audio features from the performance signal.
3. Calculation of local distances between score and performance (section 6.1).
4. Computation of the optimal alignment path which minimises the global distance (sections 6.2 and 6.3).

Only the last stage is specific to DTW. Our choice for this algorithm is due to the possibility of optimizing memory requirements. Also, unlike HMM based methods, DTW does not have to be trained, so that a hand made training database is not necessary.

For each sequence, the score and the performance are divided into frames described by features. Score information is extracted from standard MIDI files, the format of most of the available score databases. However this format is very heterogeneous and does not contain all classical score symbols. The only available features from these MIDI files are the fundamental frequencies of the notes present at any time, and note attack and end positions.

The features of the performance are extracted through signal analysis techniques using short time Fourier transformation (usually with a 4096 point hamming window, 93 ms at 44.1 kHz). The temporal resolution of the alignment is given by the performance frame rate. For instance, a hopsize of the analysis window of 256 points gives a resolution of 5.8 ms at a sampling rate of 44.1 kHz. The number of score frames is approximately the same, so that the diagonal is the ideal alignment path. Note, however, that there is an inherent imprecision, for we only know the aligned frame and not the exact aligned position within the frame. Usually, the arbitrary choice is made to take the middle of the window for the precise position (needed, for instance, to determine the exact sample where to cut a prospective synthesis unit).

The accuracy and robustness of peak structure match based dynamic time warping can be improved by the methods introduced in (Soulez, Rodet, and Schwarz 2003), especially for polyphonic music:

- Individual re-tuning of the filter bands to the maximum energy allows to precisely match notes that may deviate from their expected frequency (see section 6.1.1).
- An advanced attack detection based on the delta between local energy maxima in the tuned filters detects the attack of very weak notes within a multitude of other notes (see section 6.1.2).

Section 6.4.3 describes further improvements being worked on which are the in-frame attack detection by a specialised algorithm, and the integration of a beat tracking method to be able to align performances containing percussion.

6.1 Calculation of Local Distances

The local score–performance distance guiding the alignment is calculated for each pair of a frame m in the performance and a frame n in the score. This distance, representing the similarity of the performance frame m to the score frame n, is calculated using spectral information. The local distances are stored in the *local distance matrix* $ldm(m, n)$, see figure 6.1 for an example. Only a part of the matrix needs to be calculated, because local and global path constraints, described in sections 6.3 and 6.4, reduce the number of points (m, n) that can be part of the optimal path.

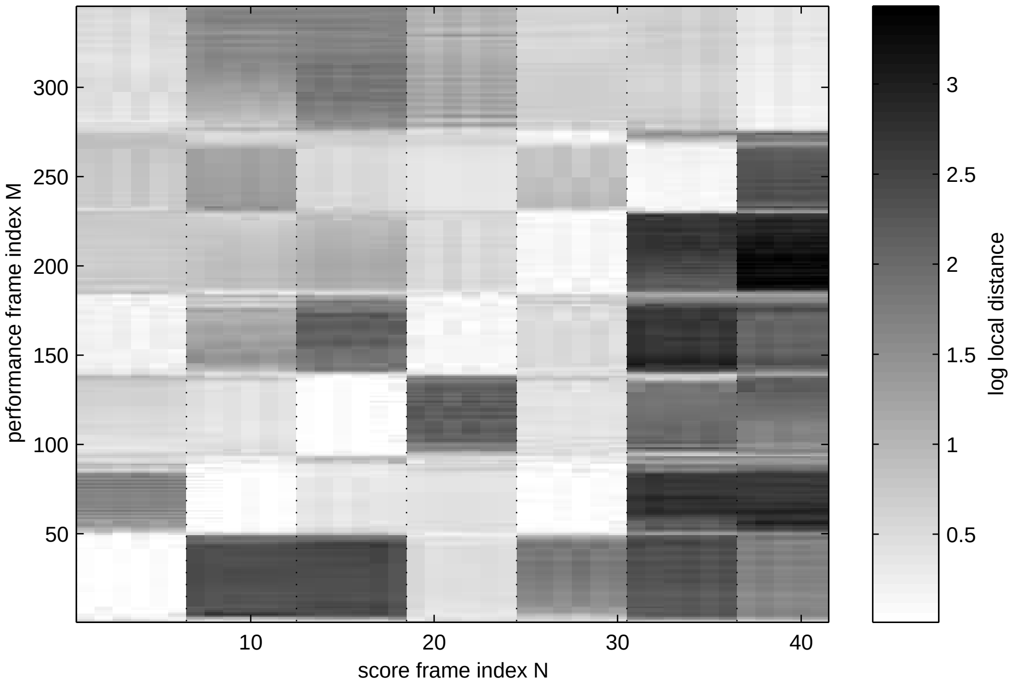


Figure 6.1: Local distance matrix of a guitar walk

The only significant features contained in the score are the pitch, the note limits and the instrument. Since having a good instrument model is difficult, only pitch and transients were chosen as features for the performance. This is why the note model is defined with attack frames using pitch and onset information, sustain frames using only pitch, and rest frames using energy.

6.1.1 Sustain Model

The sustain model uses only pitch. As pitch tracking algorithms are error prone, especially for polyphonic signals, we use the *Peak Structure Match* introduced in section 5.5. With this method, the local *Peak Structure Distance* (PSD) is the ratio of the signal energy filtered by harmonic band pass filters corresponding to each expected pitch present in the score frame, over total energy.

PSD is very efficient in monophonic, and polyphonic but mono-instrumental cases. However, for a poly-instrumental performance, the different instruments do not have the same loudness, and it is very difficult to localize low and short notes under continuous loud notes. Coding energies on a logarithmic scale reduces level ratio between the different instruments and thus improves results.

However, this model has two major drawbacks. First, in polyphonic cases, filter banks corresponding to a chord tend to cover the major part of the signal spectrum, increasing the similarity of this chord with any part of the performance. Thus, filters need to be as precise as possible.

Second, such a model with narrow filters is adapted to fixed pitch instruments, such as the piano, in which small frequency variations, error, or vibrato, are impossible. For string instruments and the voice, such variations can be as large as a halftone around the nominal frequency of the note. A simple solution is to define vibrato as a chord of the upper and the lower frequency, but vibrato is not included in most MIDI based scores. Another solution is to give a degree of freedom to each filter around its nominal frequency, as introduced in (Soulez, Rodet, and Schwarz 2003):

For each performance frame m with magnitude spectrum P, the filter is tuned within a certain range of r cents to yield the highest energy. The energy is weighted by a window (hamming or Gaussian) centered around the nominal frequency of the filter, penalising a high energy peak far away and favouring a weaker but close one.

For each filter band k, $1 \leq k \leq F_n$, of all notes p in score frame n with nominal frequency $f(n,p)$ we determine the best tuning offset t as:

$$t_{\max}(k) = \underset{-r \leq t \leq r}{\operatorname{argmax}} \left(\operatorname{hamming}(t) \sum_{i \in B_{n,p,k,t}} P_i \right) \tag{6.1}$$

with the set of bin indices of the tuned filter band given by

$$B_{n,p,k,t} = \left\{ i \in \mathbb{N} \;\middle|\; \left| i \frac{f_s}{N_{\mathrm{FFT}}} - k \cdot f(n,p) \cdot 2^{\frac{t}{144}} \right| \leq \beta_{n,p,k} \right\} \tag{6.2}$$

and the harmonic filter band width, controlled by the basic filter width given by β in tones

$$\beta_{n,p,k} = k \cdot f(n,p) \cdot 2^{\frac{1}{6}\beta} \tag{6.3}$$

Interestingly, we have observed that shifting filters independently gives better results than shifting the whole harmonic comb. This independent filter tolerance improves distance calculation for slightly inharmonic instruments, such as the piano.

This *tuned peak structure distance* (TPSD) is then used as the local distance for each sustain score frame n and performance frame m. It is defined, similarly to equation (5.4) on page 44, as:

$$TPSD(m,n) = 1 - log \frac{\sum\limits_{i \in B_{\max}} P_i}{\sum\limits_{i \in R(n)} P_i} \tag{6.4}$$

with $B_{\max}$ the union of the best tuned filter bands for this frame:

$$B_{\max} = \bigcup_{p,k} B_{n,p,k,t_{\max}(k)} \tag{6.5}$$

After a number of tests, working with the first $F_n = 6$ harmonics filters gives acceptable results. Equivalent results were obtained for $F_n = 7$ or 8. The best and most homogeneous results are obtained with a filter width of $\beta = \frac{1}{10}^{\mathrm{th}}$ semitone (10 cents) and a tolerance of about $r = \frac{3}{4}^{\mathrm{th}}$ semitones (75 cents) around the nominal frequency.

6.1.2 Attack Model

Tests using only the sustain model show some imprecision of the alignment marks, which are often late. Worse, in very polyphonic cases with more than three simultaneous notes, some notes are not detected at all.

There are three reasons for the imprecision of the markers. First, during attacks energy is often spread all over the spectrum, giving low values of the PSD because of the normalisation by total energy. Second, reverberation causes the partials of the last note to still be present at the start of the next note. Third, the energy maximum in the filters is reached several frames after the true attack, because of the sometimes slow attack of an instrument, and also because of the smearing due to windowing. With the sustain model alone, alignment marks are set at the instant when the energy of the current note rises above the energy of the last note, several hundredths of a second after the true onset.

Moreover, in the polyphonic case, during chords, several notes often have common partials. If only one note of this chord changes, too few partials may vary to cause enough difference in the spectral structure to be detectable by PSD.

However, the energy at the harmonic peaks of the expected notes rises sharply during the attack. Hence, a more accurate indication of a note beginning is given by the variation in the filters. Thus, special score frames using energy variations Δ_k^p in the harmonic filter band k of the note p instead of PSD are created at every onset. In these frames, the attack distance AD is given by the sum of the energy variations (in dB) in every tuned filter band k of the TPSD from equation (6.4). In the case of simultaneous onsets, the distance AD is computed for every beginning note and averaged out:

$$AD(m,n) = \operatorname*{mean}_p \left(1 - \tanh\left(\alpha\left(\sum_{k=1}^{F_n} |\Delta_k^p| - \theta_a\right)\right)\right) \tag{6.6}$$

with Δ_k^p the energy difference in dB with the previous local extremum in the filter band k of note p, θ_a a threshold, and α a scaling factor. Small note changes during chords seem to be grasped by human perception mostly due to their onsets. Therefore, the local distance AD is amplified by the scaling factor α to favor onset detection over PSD. After some tests, θ_a was set to 6.5 dB and α to 50.

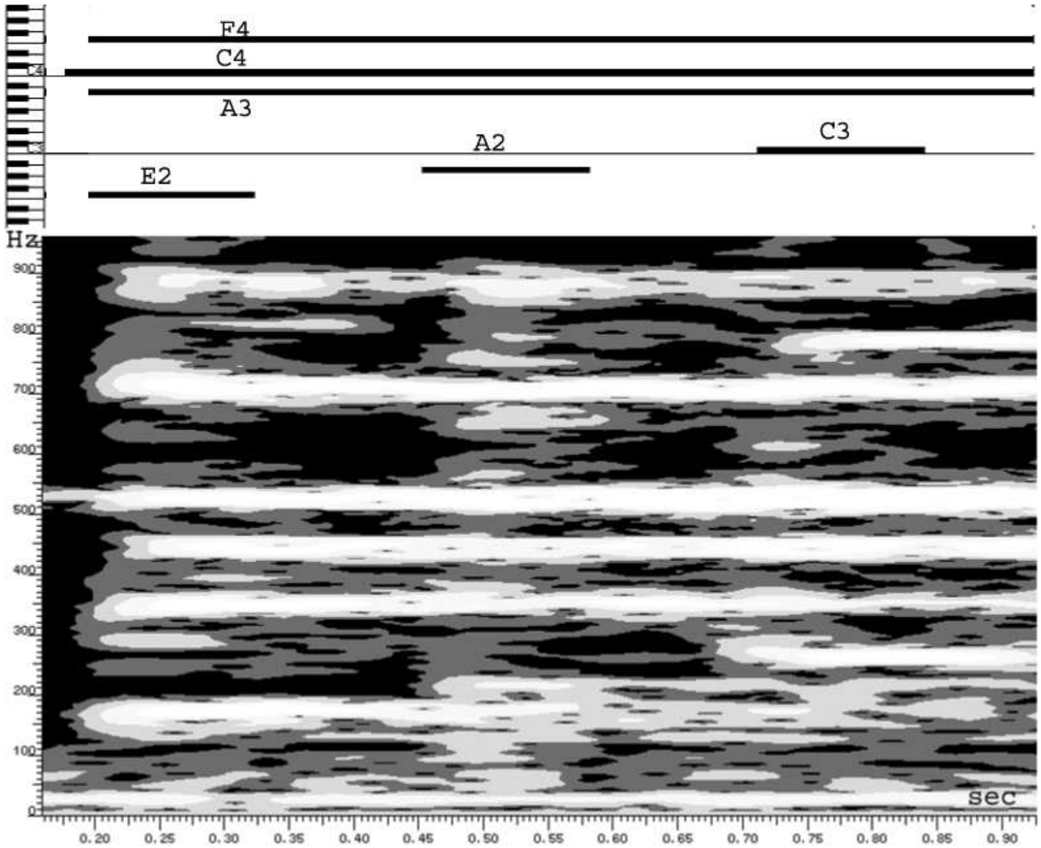
The example in figure 6.2 is characteristic of the principal problems of the sustain detection: For the first second of this Mozart string and oboe quartet, violins and oboe play a loud continuous note while the cello is playing small notes in their sub-harmonics. The cello has many common partials with the other notes and global energy variations are due to violin vibrato and not cello onset. As shown by the PSD diagram in figure 6.2(b), detection by use of the sustain model (PSD) is not possible. On the contrary, the three notes E2, A2 and C3 can easily be localized on the energy variation diagram as indicated by the vertical dash-dotted lines.

6.1.3 Silence Model

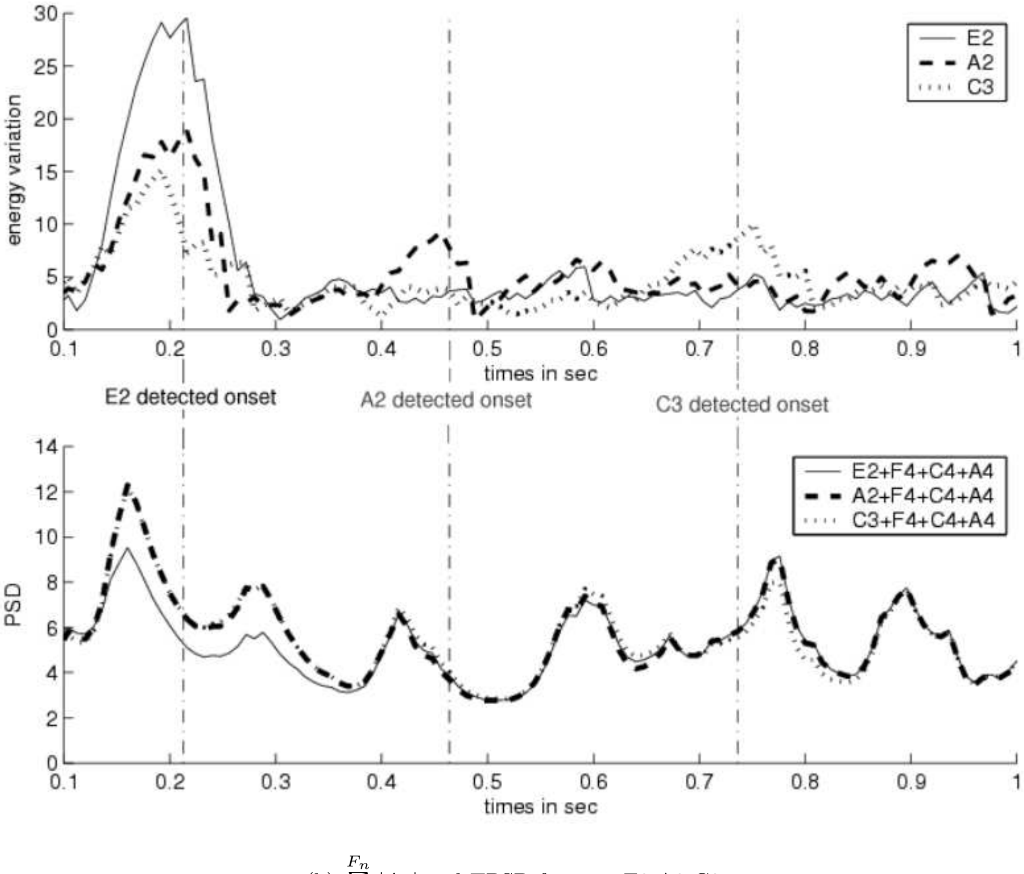
We introduce special score frames at the end of each note to correctly handle possible silence caused by rests in the score. Short silences, however, due to short rests in the score and non-legato playing are difficult to model, since reverberation has to be taken into an account. We only model rests longer than 100 ms. Shorter rests are merged with the previous note.

For these frames, a special distance SD measures the match of the signal log energy E above a silence threshold θ_s. This allows the alignment path to stay in the silence frame in the score and advance in the performance in order to "stretch out" the pauses between notes.

$$SD(m,n) = \begin{cases} E - \theta_s & \text{if } E \geq \theta_s, \\ 0 & \text{if } E < \theta_s. \end{cases} \tag{6.7}$$



(a) Spectrogram and MIDI roll.



(b) $\sum_{i=1}^{F_n} |\Delta_i|$ and TPSD for note E2 A2 C3

Figure 6.2: First second of Mozart quartet (from Soulez, Rodet, and Schwarz 2003)

where E is the total logarithmic energy of the signal in the performance frame m:

$$E = \log \sum_{i=1}^{N_{\mathrm{FFT}}} P_i \tag{6.8}$$

6.1.4 Combination of Local Distances

The three local distance models are combined to build the local distance matrix: They are used only for their specific attack, sustain, and release score frames, i.e. columns in the ldm, which is given by:

$$ldm(m,n) = \begin{cases} AD(m,n) & \text{if } n \in A \\ SD(m,n) & \text{if } n \in S \\ TPSD(m,n) & \text{otherwise} \end{cases} \tag{6.9}$$

where A and S are the first and last frames of all notes, respectively. Figure 6.3 shows how the three different score frame types are combined to calculate the local distances.

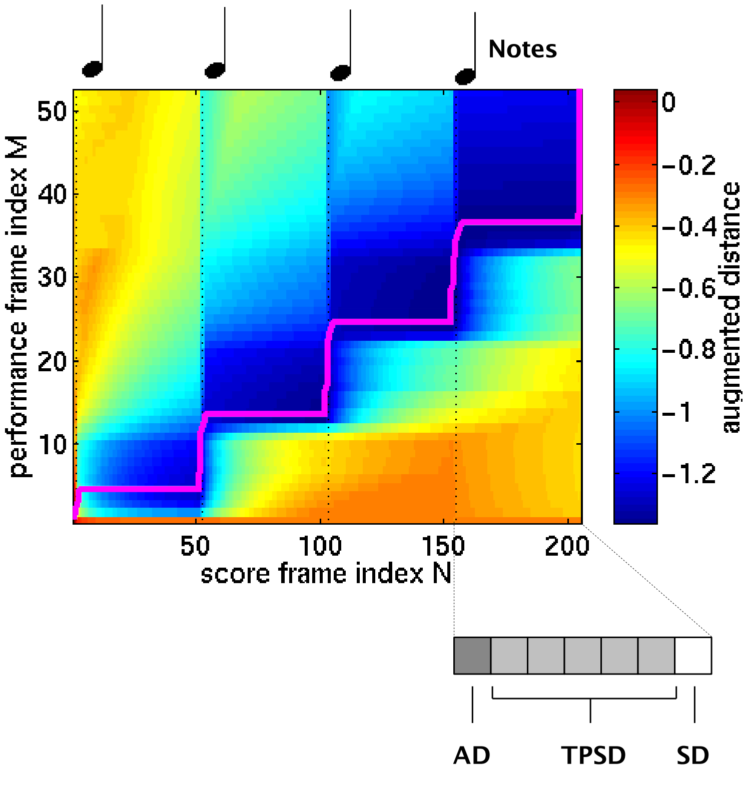


Figure 6.3: Use of the different score frame types for alignment by DTW

6.2 Local Path Constraints

The local distances and the local and global constraints are used by the DTW algorithm to calculate an *augmented distance matrix* $adm(m,n)$ which is the cost (the accumulated total distance) of the best path up to the point (m,n). See figure 6.4 for an example. The alignment is given in the form of a path through the augmented distance matrix. If a path goes through (m,n), the frame m of the performance is aligned with frame n of the score.

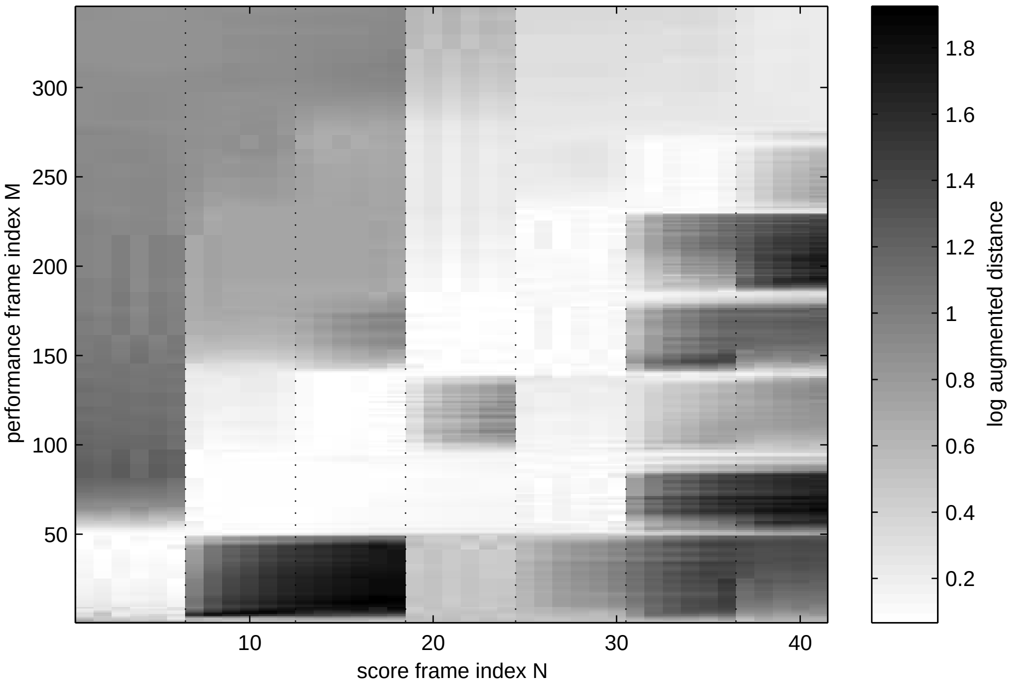


Figure 6.4: Augmented distance matrix of a guitar walk with alignment path

The three different local path constraints, also called *neighbourhoods*, that were tested are the types I, III, and IV, according to the terminology in (Rabiner and Juang 1993). Type I, the simplest one, gives satisfactory results for mainly monophonic music, as is useful for unit database building. However, types III and IV are more robust to mismatches between the score and the performance, for instance missing or inserted notes or groups of notes, or differences in interpretation.

The weights attached to the local path constraint branches are w_v for the vertical, w_h for the horizontal, and w_d for the diagonal, as shown in figure 6.5. They can be tuned in order to favour one direction.

The local path constraints define how the augmented distance matrix adm at point (m, n) is calculated from the local distances, with $ldm(m, n)$ abbreviated to λ:

Type I:

$$adm(m,n) = \min \left\{ \begin{array}{ll} adm(m-1,n-1) & + w_d\,\lambda \\ adm(m-1,n) & + w_v\,\lambda \\ adm(m,n-1) & + w_h\,\lambda \end{array} \right\} \tag{6.10a}$$

Type III:

$$adm(m,n) = \min \left\{ \begin{array}{l} adm(m-1,n-1) + w_d\,\lambda \\ adm(m-2,n-1) + w_v\,\lambda \\ adm(m-1,n-2) + w_h\,\lambda \end{array} \right\} \tag{6.10b}$$

Type V:

$$adm(m,n) = \min \left\{ \begin{array}{l} adm(m-1,n-1) + w_d\,\lambda \\ adm(m-2,n-1) + w_v\,\lambda + w_d\,ldm(m-1,n) \\ adm(m-1,n-2) + w_h\,\lambda + w_d\,ldm(m,n-1) \\ adm(m-3,n-1) + w_v\,\lambda + w_d\,ldm(m-2,n) + w_v\,ldm(m-1,n) \\ adm(m-1,n-3) + w_h\,\lambda + w_d\,ldm(m,n-2) + w_h\,ldm(m,n-1) \end{array} \right\} \tag{6.10c}$$

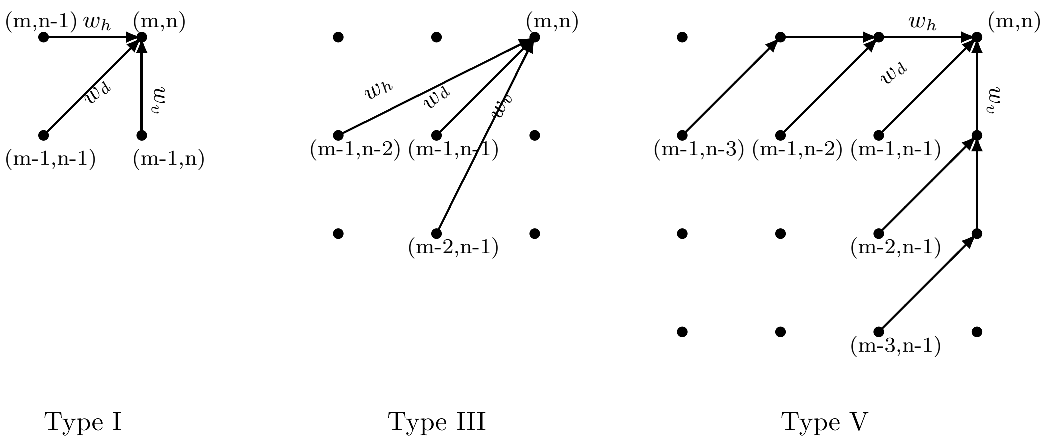


Figure 6.5: Neighbourhood on point (m,n) in type I, III and V

The constraint type I is the only one allowing horizontal or vertical paths and thus admitting extra or forgotten notes. The drawback of this constraint type is that the path can get stuck in a "valley" along the score axis with erroneous small local distance with the current performance frame. This can lead to bad results for polyphonic scores by detecting too many extra or forgotten notes.

The types III and V constrain the slope to be respectively between 2 and $\frac{1}{2}$ or 3 and $\frac{1}{3}$, i.e. the path is forced to advance in both directions. Since it is very rare for a performance to contain passages played more than three times faster or slower than the score, it gives a good alignment but will accept neither vertical nor horizontal paths and thus does not directly handle forgotten or extra notes.

These constraints III and V give approximately the same result, the type V takes more resources and more time but gives more freedom to the path allowing greater slope. Using Type V is preferable but type III can still be used for long pieces.

The standard values for the local path constraint weights $(w_v, w_h, w_d) = (1, 1, 2)$ for type I and V or $(3, 3, 2)$ for type III, do not favour any direction and are used in our method. Note that our experiments showed that lowering w_d favours the diagonal and prevents extreme slopes, which means that music where the tempo does not fluctuate too much is better aligned.

6.3 The DTW Algorithm

DTW is based on the Viterbi algorithm (Viterbi 1967; Forney 1973), which is an instance of the class of matrix-based $O(n^2)$ algorithms called *dynamic programming*.

The DTW algorithm finds the best alignment path between two sequences according to local distances and a number of local and global constraints. The local distances are stored in the local distance matrix, see section 6.1, where each value $ldm(m, n)$ expresses the dissimilarity between the score frame m and the performance frame n.

Additionally to the local path constraints from above, the following global constraints have been applied: The end points of the alignment path are set to be $(1, 1)$ and (M, N), where M and N are the number of frames of the performance and of the score, respectively. The path is monotonic in both dimensions. The score is stretched to approximately the same duration as the performance $(M \approx N)$. The optimal path should then be close to the diagonal, so that favouring the diagonal would prevent deviating paths.

The best path is computed iteratively, by updating the augmented distance matrix $adm(m, n)$, which is the cost of the best path up to the point (m, n). The matrix $\psi(m, n)$ keeps the backpointer of the path to the previous point. For simplicity, we show here the algorithm for type I. The other types differ only in the calculation of $adm(m, n)$ using one of the equations 6.10 and in the boundary condition initialisation.

Starting from the initial conditions, we initialise the boundaries as follows:

$adm(1, 1) = ldm(1, 1)$
$\psi(1, 1) \quad = (0, 0)$
for $2 \leq n \leq N$:
  $adm(1, n) = adm(1, n-1) + w_h\, ldm(1, n)$
  $\psi(1, n) \quad = (1, n-1)$

And then calculate performance frame by performance frame:

for $2 \leq m \leq M$:
  $adm(m, 1) = adm(m-1, 1) + w_v\, ldm(m, 1)$
  $\psi(m, 1) \quad = (m-1, 1)$
  for $2 \leq n \leq N$:

$$adm(m, n) = \min \begin{Bmatrix} adm(m-1, n-1) + w_d\, ldm(m, n) \\ adm(m-1, n) \qquad + w_v\, ldm(m, n) \\ adm(m, n-1) \qquad + w_h\, ldm(m, n) \end{Bmatrix}$$

    set $\psi(m, n)$ to the chosen predecessor point with minimum distance

The decoding of the path matrix ψ to find the optimal alignment path π is done in reverse order by following the backward indices:

$\pi = [\,]$
$p = (m, n)$
while $p \neq (0, 0)$:
    $\pi = [p, \pi]$ *(prepend current point p to π)*
    $p = \psi(p)$

6.4 Improvements of DTW

Several improvements have been added to the classical DTW algorithm in order to lower processing time or memory requirements and thus allow long performances to be analysed. The most important of these improvements are the path pruning and the shortcut path implementation

6.4.1 Path Pruning

With a typical frame hop size of 5.8 ms, a three minute long performance contains about 36000 frames, so that about $1.3 \cdot 10^9$ elements need to be computed in the local distance matrix and as many for the augmented distance matrix. To reduce the computation time and the resources needed, at every iteration m, only the best paths are kept, by pruning the paths with an augmented distance $adm(m, n)$ over a threshold θ_P. This threshold is dynamically set using the minimum of the previous adm row. After various experiments this threshold was set to:

$$\theta_P(m) = 1.1 \cdot \min\Big(adm(m-1)\Big) \tag{6.11}$$

However, the paths between the corridor of selected paths and the diagonal are not pruned to leave more possible paths. Usually the corridor width is about 400 frames.

6.4.2 Shortcut Path

The DTW algorithm can be implemented efficiently such that the performance need not be present in memory as a whole. Equally, the distance matrices are accessed only in the neighbourhood of the current performance frame, so that only the last two lines of these need to be kept in memory for local path constraint type I, up to 4 for type V. However, the matrix ψ that stores the pair of least cost path indices from each point (m, n) can not be reduced, because we only know at the end which path is the globally optimal one.

This poses memory problems for the fully automated application of DTW on real-world sound files. As an example, the first movement of the Sonata 1 for solo violin by J.S. Bach lasts $2\frac{1}{2}$ minutes and contains $N_n = 450$ notes. This yields about $M = 24000$ frames for the performance and the score, and a path matrix of around $2M^2 = 1.152 \cdot 10^9$ elements for ψ, taking up at least 4 GB of memory. A global path constraint, i.e. considering only a central corridor for the possible paths, reduces the memory requirement by only a factor of 2, leaving it still too high for today's computers.

However, all we are interested in is the alignment of note onset times. We don't care about (and indeed didn't model) the evolution within a note. This means that we only need to keep in memory all possible *shortcut paths*, as presented in (Orio and Schwarz 2001), i.e. paths that are reduced to the first and last score frame for each note. Their memory requirement is only $2MN_n = 21.6 \cdot 10^6$ elements, i.e. around 80 MB, which means a reduction by 98% for the path matrix.

6.4.3 Further Possible Improvements

Integrating a transient or onset detection algorithm such as (Rodet and Jaillet 2001; Roebel 2003; Hainsworth and Macleod 2003) allows to overcome the intrinsic in-frame imprecision by redetecting the note onset within the aligned frame. Preliminary results showed an improvement of 3.5 ms for a corpus of synthesised test performances.

The shortcoming of being limited to music without drums or percussion can be overcome by combining score and beat alignment: Beat tracking by iterative refinement of prototype percussion sounds (Gouyon 2000) is integrated at the score frames where percussion sounds are expected. There, the correlation-based distance of the sound around the frame to percussion sound classes is used as distance instead of AD or $TPSD$.

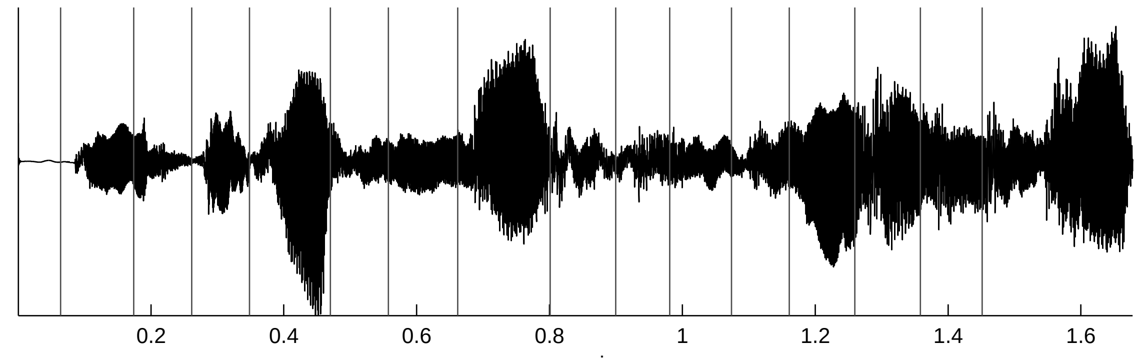
6.5 Results of DTW Alignment

Due to the absence of aligned reference corpora and the difficulty of building one by manual alignment, quantitative test with result statistics were done on synthesised performances (section 6.5.4) and a small corpus of recorded performances (section 6.5.5).

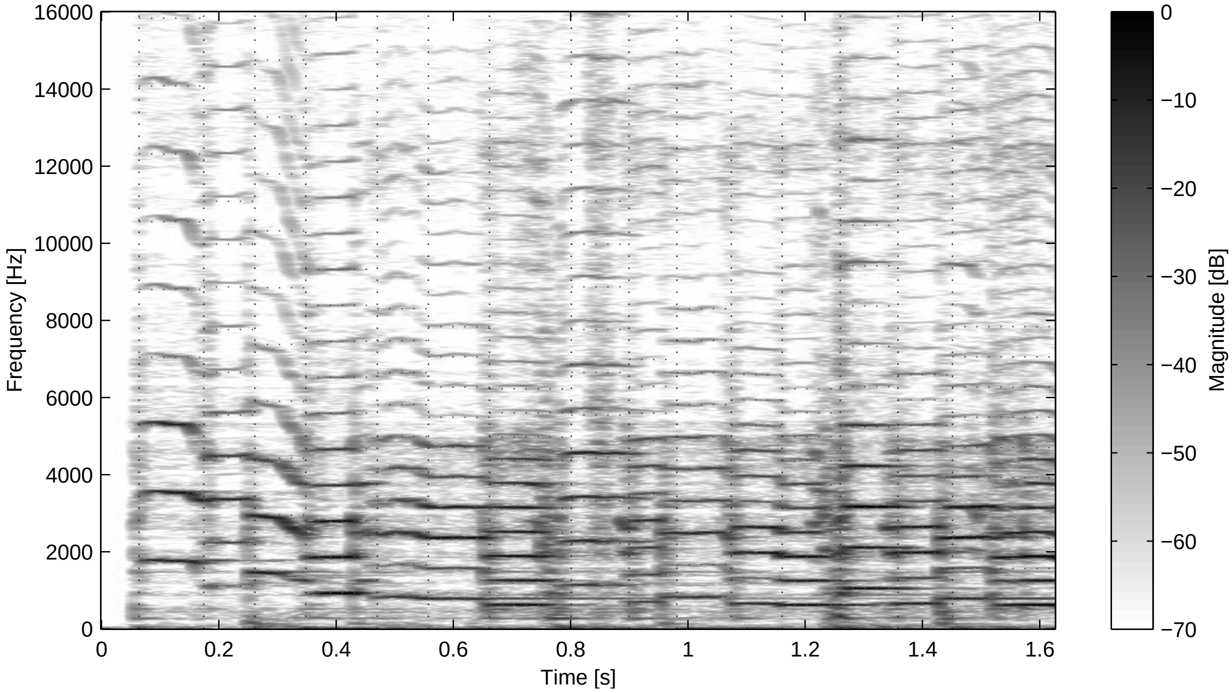
However, many qualitative tests were performed by listening to performances and their reconstituted MIDI files, which permitted the evaluation of global alignment. These tests were performed with various types of music, (classical, contemporary, songs without percussion, for instance J. S. Bach, W. A. Mozart, Fréderique Chopin, Pierre Boulez, Georges Brassens, etc.).

Even very difficult signals, such as held chords with only one changing note, the singing voice, very fast piano or violin passages (e.g. 16 legato notes at a rate of 10 notes per second with irregular accents in figure 6.6), and performances with trills and vibrato (figure 6.7) were perfectly aligned. Tests on multi-instrument music (a string and oboe quartet in figure 6.8) showed a good global alignment with only few imprecisions in the determination of the note onsets.

All tests were performed with a default frame hop size of 5.8 ms (usually 256 points) which is a good compromise between precision and number of frames to compute. This hop-size can be lower for a better resolution when considering small recordings or higher for quick preview of the alignment.

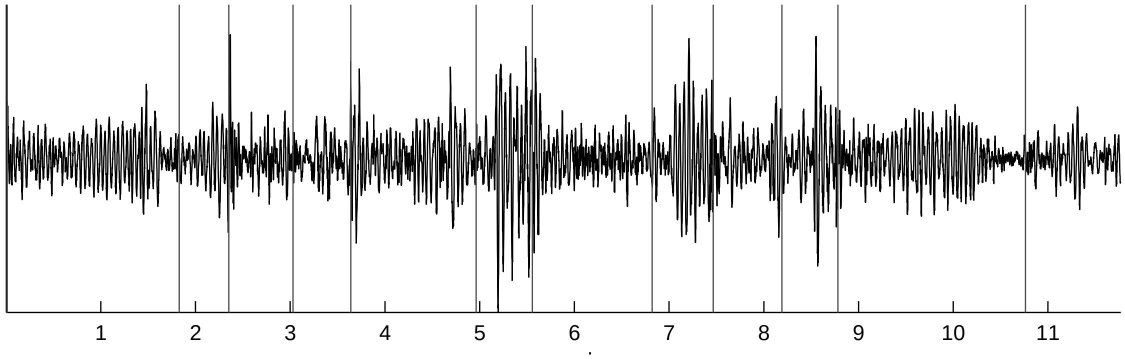


(a) Waveform and alignment marks

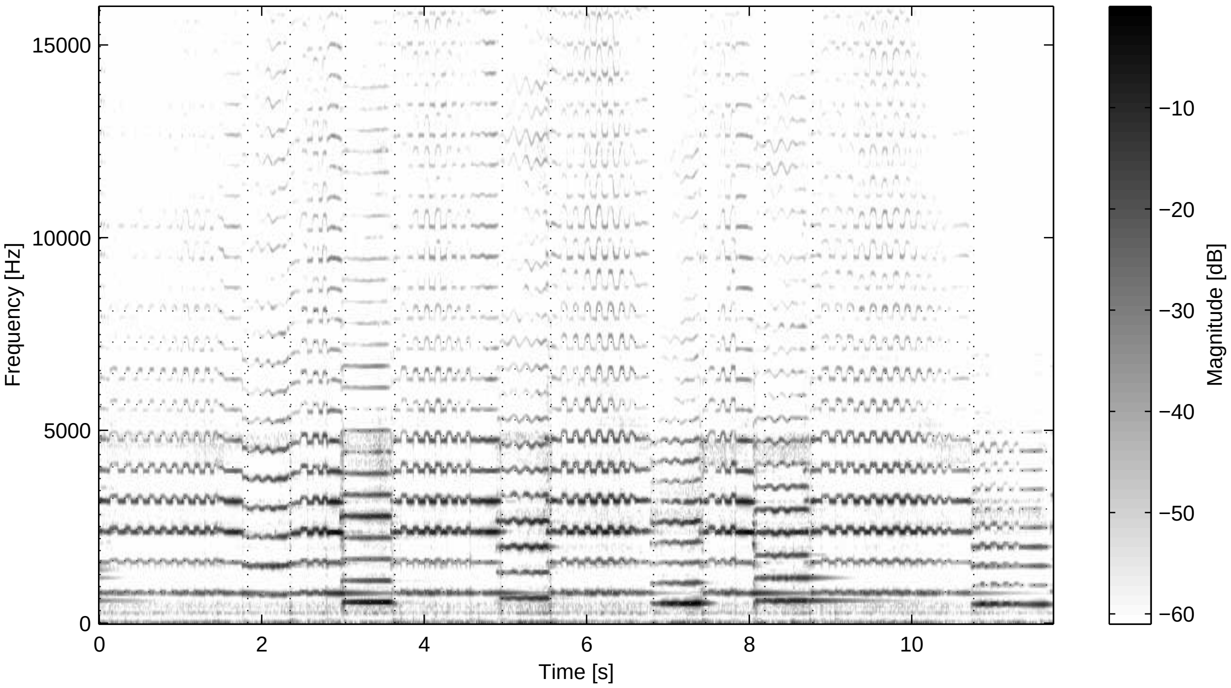


(b) Spectrogram and alignment marks

Figure 6.6: DTW Alignment result for a fast excerpt of the introduction of *Anthèmes 2* by Boulez

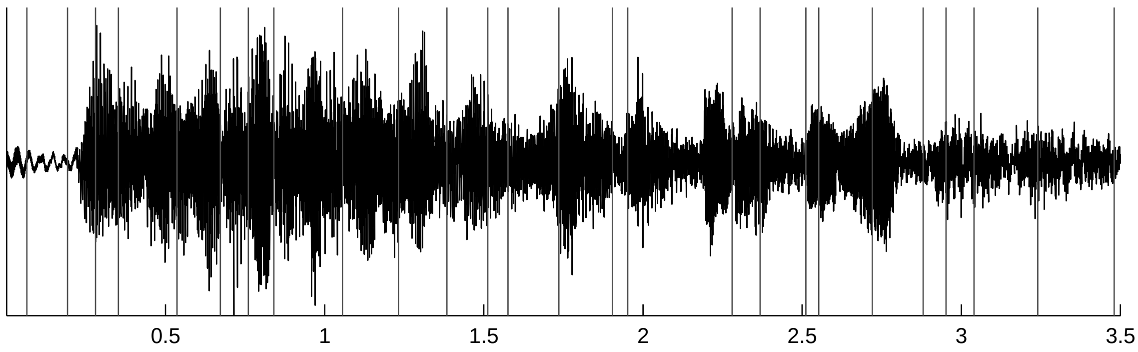


(a) Waveform and alignment marks

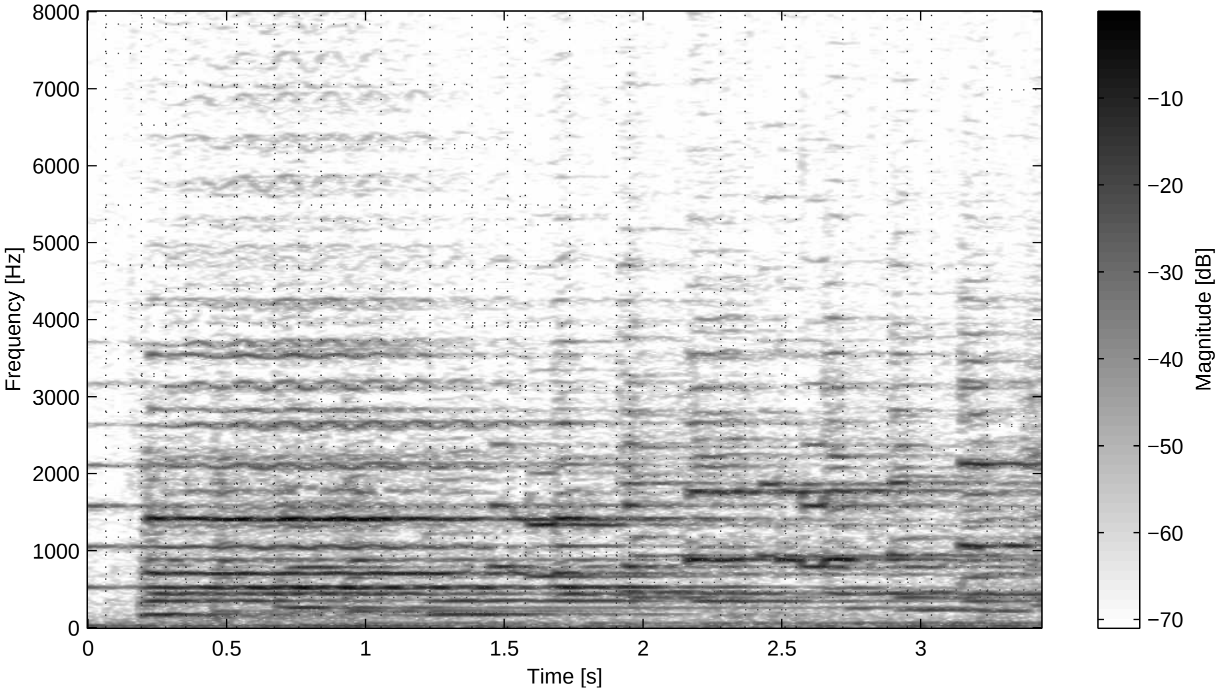


(b) Spectrogram and alignment marks

Figure 6.7: DTW Alignment result for an excerpt with trill of *Anthèmes 2* by Boulez



(a) Waveform and alignment marks



(b) Spectrogram and alignment marks

Figure 6.8: DTW Alignment result for an excerpt of the *Strings and Oboe Quartet* by Mozart

6.5.1 Limitations

Notes shorter than 4 frames (23 ms) are very difficult to detect and often lead to errors for neighbouring notes. Therefore, all the score states that are too short are merged in a chord with the next state. This technique makes it possible to handle unquantised chords from MIDI files recorded on a keyboard.

Alignment is efficient for pieces with less than five harmonic instruments. As the memory requirement is still too high, only pieces shorter than about ten minutes and with about four thousand or less score events are currently treatable (a little less with local constraint V), but this is enough to align most pieces. The most copious successful test (with the highest number of events) was performed on a five minute and twelve second long jazz performance of 4200 score events with time resolution of 5.8 ms (53926 frames) taking about 400 MB of RAM and 146 minutes on a Pentium IV 2.8 GHz running C++ and MATLAB routines.

The longest sucessfully aligned sound file so far is the second movement of the *Sonata No. 2* for solo violin (7 minutes, 1703 events) which needed 900 MB of memory and ran in just over 2 hours with a time resolution of 5.8 ms on a machine with 1 GB of RAM. On 512 MB of RAM plus 1 GB of swap space, it took 3 days due to extensive swapping, on less memory than that, alignment doesn't complete at all. This is due to the quadratic space complexity of alignment for the shortcut path matrix (section 6.4.2) which takes $O(M, N_n)$ frames, with M the number of performance frames, and N_n the number of notes in the score.

6.5.2 Alignment Quality Indicator

As performers rarely play with sudden variations in tempo, extreme slopes of alignment path, with large variation, usually indicate score–performance mismatching. Thus, the path slope can be a good error indicator. If the slope is $\frac{1}{3}$ for several notes, it is very likely that some notes are missing in the performance. On the other hand if the slope is 3, there are certainly extra notes in it.

This indicator was able to find with precision the position of an unknown extra measure in a MIDI file of J. S. Bach's first piano prelude, as can be seen in figure 6.9.

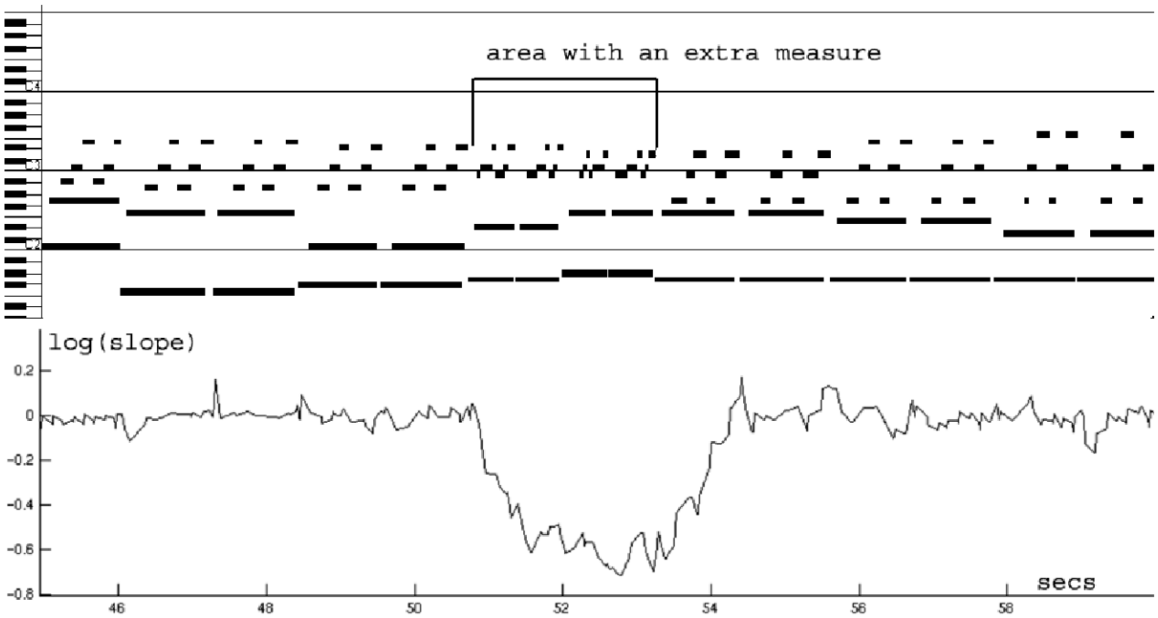


Figure 6.9: Piano roll representation of aligned MIDI, and path slope in log units in the Bach prelude between 45 sec and 60 sec (from Soulez, Rodet, and Schwarz 2003).

6.5.3 Robustness

Tests with audio recording that do not exactly coincide with the MIDI files showed very strong robustness and a very good global alignment. For instance, alignment of the first prelude for piano of J. S. Bach (80 s and 629 score events) with an extra measure at the 51st second was correctly aligned until the 50th and after the 55th, and another test with a Bach sonata for violin showed a very good global alignment even though a passage of 52 notes was missing in the score! Vibratos and trills can be aligned very efficiently as well, as shown in the very large section with trills of *Anthèmes 2* by Boulez in figure 6.7.

6.5.4 Tests on Synthesised Performances

Quantitative tests have been carried out on 708 performances played by a sample based synthesizer. The choice of synthesized performances provides a precise reference of note onsets in the performance, without requiring a manual alignment. We used 14 different sounds, played with the 4 different levels of articulation *legato*, *detaché*, *pause*, and *staccato* (see Takala, Hiipakka, Laurson, and Välimäki 2000), with gradually longer gaps between the notes. We prepared six different scores, composed by Nicola Orio, three monophonic, two with two voices of polyphony, and one with three voices of polyphony. The piano-roll representation of three of these score is shown in figures 6.10– 6.12. Among the monophonic scores, one is a simple repetition of the same note, and one is deliberately mismatched with its performances by one octave to test the robustness of the algorithm. The scores were played at 3 different transpositions, each 2 octaves apart.

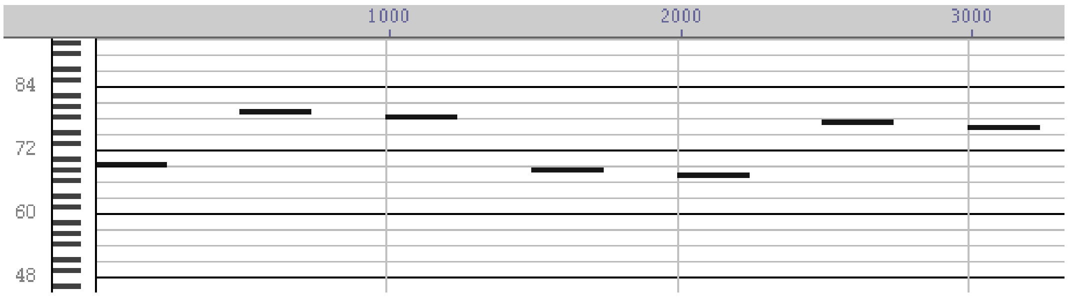


Figure 6.10: Pianoroll of score *walk*

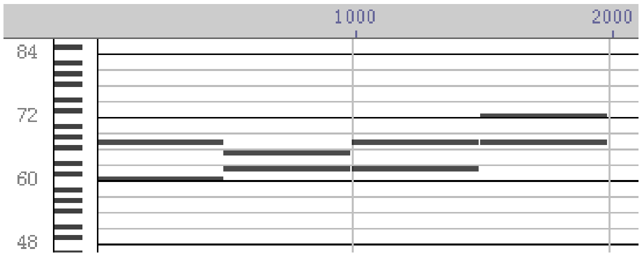


Figure 6.11: Pianoroll of score *bichord*

We had to eliminate the results given by one of the sounds, because it was a bell whose inharmonic spectrum is not modeled by our technique. The results for the octave-mismatched score were encouraging, but the robustness on octave deviations has still to be extensively tested. As we expected, our technique is not very suitable for the performances with repeated pitch, because the peak structure does not change between notes. The technique needs to be improved by using other features for dealing with this special case.

In the following sections we present the quantitative analyses on the remaining 480 examples.

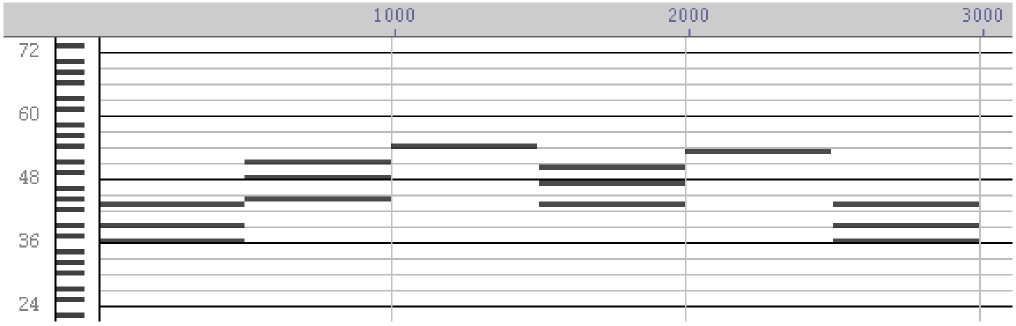


Figure 6.12: Pianoroll of score *trisingle*

6.5.4.1 Error Rate

We considered an error an alignment mark more than 200 ms off of the expected performance position. Of all the files, only 9.4% had erroneous marks at all (11.9% without the attack and silence modeling). For monophonic performances, this rate drops to 2.5%. The percentage of alignment errors over all marks in all performances is 2.5% (0.42% for the monophonic, 3.6% for the polyphonic performances).

The error rate is lowest for the middle octave, and drops with the introduction of longer pauses. However, for the staccato playing style, we noticed a small increase of the error rate. The same effect was noticed when only the PSD was used.

6.5.4.2 Offset

A more detailed parameter is the average offset (i.e. the absolute distance between the expected and found alignment mark) on non erroneous marks.

As can be seen in table 6.1, there is a decrease of the offset for higher octaves, due to the larger window size needed to resolve low frequency spectral peaks. Moreover, the offset generally decreases with longer pauses, with the exception of the lowest octave.

articulation	low	mid	high	avg
legato	44	33	26	34
detaché	20	16	8	15
pause	35	11	8	18
staccato	37	10	9	19
avg	34	18	13	**23**

(a) Results for monophonic scores

articulation	low	mid	high	avg
legato	58	36	35	43
detaché	26	18	15	20
pause	33	13	7	18
staccato	35	10	9	18
avg	38	19	16	**26**

(b) Results for polyphonic scores

Table 6.1: Average offset in ms depending on articulation and octave.

Regarding the comparatively high offsets for legato articulation, listening to the found segments revealed that the algorithm chose to place the note onset where the overlapping partials of the previous note had sufficiently died down, which is actually better suited for building unit databases.

When the alignment is computed without the attack and silence modeling, the average offset is 31 ms, which if compared to the total average of the complete modeling of 25 ms, justifies its higher complexity.

6.5.5 Tests on Jazz Piano recordings

Quantitative tests were performed on several jazz piano improvisations, played by four different musicians, where sound and MIDI were both recorded. These are very fast (an attack every 70 ms on the average) and long pieces (about four minutes) with many trills and a wide dynamical range.

As reverberation prevents precise note end determination, we focused on note onset detection. Only a good global alignment was looked for. A correct pairing between score and performance means that the detected note onset is closer to its corresponding onset in the performance than any other. With this criteria, tests showed a 9.7% error rate of onset detection over the 9024 considered onsets, about 65% of these errors were made on notes shorter than 80 ms, coresponding to a rate of 12 notes per second. These results need several comments:

- Due to the MIDI recording system used, the MIDI file, though recorded from the keyboard simultaneously with the audio seems to be relatively imprecise when compared to the audio.
- During the MIDI parsing, every note shorter than 4 frames (usually 23 ms) is merged with the preceding note, increasing error rate of small notes (numerous in our tests).
- The hop size gives 5.8 ms maximum resolution between each possible detection.
- Finally, as audio features are extracted from a short time fast Fourier transform computed on a 93 ms (4096 points) window, the center of this window is taken to determine frame position in the recording. A better solution would be to take the center of gravity of energy in this window, but this function is not yet implemented.

As a consequence, tests showed a 23.8 ms standard deviation between the score onset and the detected one. This result can easily be improved in the near future, by a second stage of precise time alignment within the vicinity of the alignment mark. In fact, on other examples this was recently reduced to 12 ms.

Chapter 7

Hidden Markov Models

In the last chapter we have seen how to use dynamic time warping to perform automatic music alignment. Another method is to use *Hidden Markov Models* (HMMs), which model a non-stationary stochastic process.

After a brief presentation of the basics of HMMs in section 7.1, we explain how they are used for real-time score following in section 7.2, which is the bases for their use in music alignment described in section 7.3, and show results in section 7.5.

7.1 Basics of Hidden Markov Models

Hidden Markov Models are extremely popular for all domains where a trainable model of the process that is analysed, exists. For more details, see the quick introduction by Roweis (1997), the classic tutorial by Rabiner (1989), or the profound elaboration with application to speech recognition by Rabiner and Juang (1993).

HMMs are essentially stochastic finite state automata with N possible states $S = \{s_1, s_2 \ldots, s_N\}$, which emit a symbol or a vector (a continuous multidimensional value) $o(t)$ each time step t a state s_i is entered. The emission is from an alphabet V and determined by a probability density function (PDF) $b_i(o(t))$. The sequence of generated emissions, which are called *observations*, of length T is $O = o(1)o(2) \ldots o(t)$. The transition probability between states s_i and s_j is given by the transition matrix a_{ij}. The transitions with non-zero probability define the topology of the model.

A specific Hidden Markov Model λ is then defined by a tuple of (S, V, a, b). The joint probability that a model λ generated an observation sequence O with a state sequence $Q = q(0)q(1) \ldots q(t)$, where $q(0)$ is a non-emitting initial state, is given by

$$P(O, Q|\lambda) = a_{q(0),q(1)} \left(\prod_{t=1}^{T-1} b_{q(t)}(o(t)) a_{q(t),q(t+1)} \right) b_{q(T)} \tag{7.1}$$

In practice, only the observations are known, not the state sequence, that's why the Markov model is called “hidden”. This means, only λ and O are known, and we seek the probability $P(O|\lambda)$ that the model λ generated the observations O of length T. To avoid the combinatorial complexity of calculating and summing the probabilities of every possible state sequence of length T, we introduce the *forward variable* α as the probability of being in state s_j at time step t after observing the given observations

$$\alpha_j(t) = P(o(1)o(2) \ldots o(t), q(t) = s_j|\lambda) \tag{7.2}$$

This value can be efficiently computed recursively starting from

$$\alpha_j(1) = a_{0j} b_j(o(1)), \qquad 1 \leq j \leq N \tag{7.3}$$

by the induction

$$\alpha_j(t+1) = \left(\sum_{i=1}^{N} \alpha_j(t) a_{ij}\right) b_j(o(t+1)), \qquad 1 \le t \le T-1, \quad 1 \le j \le N \tag{7.4}$$

yielding as answer

$$P(O|\lambda) = \sum_{j=1}^{N} \alpha_j(T) \tag{7.5}$$

This probability is used when HMMs do recognition: Given an observation sequence, several models λ_k are tried. The model that best explains the observations has the highest $P(O|\lambda_k)$.

For alignment, our aim is to reveal the hidden state sequence. This process is called *decoding*. We need to know the exact state sequence Q that explains best a given observation sequence O. This is done by the Viterbi algorithm (Viterbi 1967; Forney 1973), see also section 6.3.

7.2 Hidden Markov Models for Score Following

Alignment by HMM uses the results of the work on real-time score following initiated by Nicola Orio described in (Orio and Déchelle 2001; Schwarz and Orio 2002; Orio, Lemouton, Schwarz, and Schnell 2003). Score following is the synchronisation of a computer with a performer playing a known musical score (figures 7.1 and 7.2). It now has a history of about twenty years as a research and musical topic, and is an ongoing project at Ircam (ATR 2003).

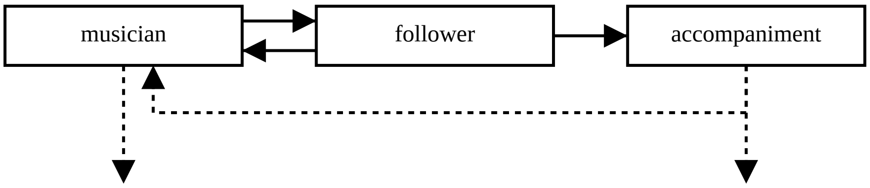


Figure 7.1: Elements of a score following system. Dashed arrows represent sound.

In order to transform the interaction between a computer and a musician into a more interesting experience, research for a virtual musician has as goal to simulate the behaviour of a musician playing with another, to create a virtual accompanist or “synthetic performer”, that will follow the score of the human musician. Score following is often addressed as “real-time automatic accompaniment”. This problematic is well defined in (Dannenberg 1984; Vercoe 1984; Vercoe and Puckette 1985), where we can find the first use of the term “score following”. Since the first formulation of the problem, several solutions have been proposed, some academic, others in commercial applications. See for instance (Baird, Blevins, and Zahler 1990; Baird, Blevins, and Zahler 1993; Bryson 1995; Dannenberg and Mont-Reynaud 1987; Dannenberg and Mukaino 1988; Grubb and Dannenberg 1994; Grubb and Dannenberg 1997; Grubb and Dannenberg 1998; Puckette 1990; Puckette 1995; Puckette and Lippe 1992; Raphael 1999b; Raphael 2001a; Raphael 2001c; Orio and Déchelle 2001; Orio, Lemouton, Schwarz, and Schnell 2003; Izmirli, Seward, and Zahler 2003).

There are a number of advantages in using a statistical system for score following, regarding the possibility of training the system and modeling different acoustic features from examples of performances and score. In particular, a statistical approach to score following can take advantage from theory and applications of Hidden Markov Models (HMMs). In fact, HMMs can deal with the several levels of unpredictability typical of performed music and they can model complex features, without requiring preprocessing techniques that are prone to errors like any pitch detectors or midi sensors. For instance, in our approach, the whole frequency spectrum of the signal is modeled. Finally, techniques have been developed for the training of HMMs.

In the context of our HMM score follower, training means adapting the various probabilities and probability distributions governing the HMM to one or more example performances such as to

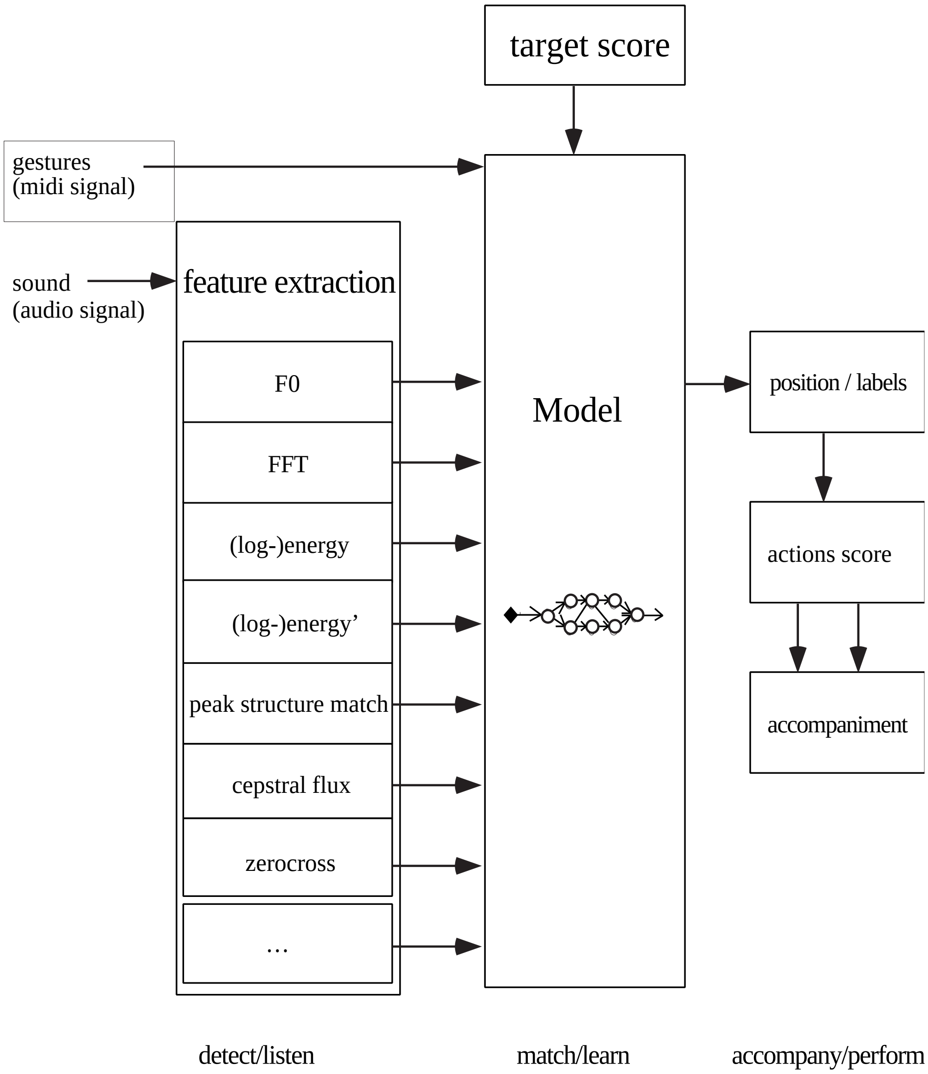


Figure 7.2: Structure of a score follower (from Orio, Lemouton, Schwarz, and Schnell 2003)

optimise the quality of the follower. At least two different things can be trained: the transition probabilities between the states of the Markov chain (Orio and Déchelle 2001), and the probability density functions (PDFs) of the observation likelihoods. While the former is applicable for audio and Midi, but needs much example data, especially with errors, the latter can be done for audio by a statistical analysis of the features to derive the PDFs, which essentially perform a mapping from a feature to a probability of attack or sustain or rest.

Then of course a real iterative training (supervised by providing a reference alignment, or unsupervised starting from the already good alignment to date) of the transition and observation probabilities is possible to increase the robustness of the follower even more. This training can adapt to the "style" of a certain singer or musician.

7.3 Hidden Markov Models for Alignment

To use Hidden Markov Models for alignment, we identify the observation sequence with the features extracted from the performance (section 7.3.1) and the state sequence with the score. Orio and Déchelle (2001) introduced a two-level model, which distinguishes between a low-level note model (section 7.3.2), and a high-level score model (section 7.3.3). They also proposed a real time decoding algorithm described in section 7.3.4, differing from the standard HMM decoding, that tells us which HMM state is aligned with the current performance frame.

We use the HMM system developed for score following by Nicola Orio (Orio and Déchelle 2001) for alignment by writing the recognised score events to an SDIF file at the performance time as marker information (section E.1). Other work on alignment using HMMs is (Loscos, Cano, and Bonada 1999b; Loscos, Cano, and Bonada 1999a; Orio and Déchelle 2001; Shalev-Shwartz, Dubnov, Friedman, and Singer 2002).

7.3.1 Signal Analysis

The features (or *descriptors*) extracted from the performance signal are listed in the following. See chapter 10 for the details of how to compute them.

Peak structure match (psm)
The main feature is the peak structure match (PSM), defined in section 5.5, which is a normalised measure of the signal energy around the harmonic peaks of the expected notes.

Delta peak structure match (dpsm)
The derivative of the PSM helps detecting a note attack faster.

Logarithmic energy (len)
The logarithmic energy serves to distinguish pauses from notes.

Delta logarithmic energy (dle)
The derivative of the logarithmic energy gives a fast indication of a note attack, even with transient noise where the PSM does not yet match

Cepstral difference (cpd)
The cepstral difference (see below) was introduced for the singing voice to detect repeated notes with the same pitch where only the vowel changes.

Zero crossing rate (zcr)
To detect pitch-less fricatives in the singing voice, the zero crossing rate gives an indicator of the noisiness of the signal.

The *cepstral difference*, or *cepstral flux*, shows the amount of change in the cepstrum spectral envelope (Schwarz and Rodet 1999; Schwarz 1998; Oppenheim and Schafer 1975). It is defined as

$$cpd = \sum_{i=1}^{R} (c_i - c'_i)^2 \tag{7.6}$$

where R is the order of the cepstrum, here 12, c and c' are the vectors of the current and the previous cepstral coefficients, respectively, calculated from a window of the signal S by

$$c = \mathrm{IFFT}\left(\log\left|\mathrm{FFT}(S)\right|\right) \tag{7.7}$$

The magnitude term $|\mathrm{FFT}(S)|$ is already calculated for the PSM, so only the inverse FFT calculation is added.

7.3.2 Note Model

The low level models a note of the performance. Each model is a left-to-right HMM consisting of n states with self-transition probabilities p for the sustain states (figure 7.3). The release state can be skipped to accomodate legato playing. The sustain states are *tied*, i.e. they share the same observation, and are trained together. Their number n models the duration of the note (Mouillet 2001). Usually, this is done with only one sustain state and the self-transition probability, e.g. in

(Loscos, Cano, and Bonada 1999b), which models less precisely the expected duration of the note given by the score.

We call the states of the note models *low-level states* to distinguish them from the states in the high-level model. The boxes in the figures represent entry and exit pseudo states that have no observation. They are used to link low-level models together. In the implementation, they serve as an "accumulator" for the total probabilities to enter a note, or to leave a note.

The note model has been extended as shown in figure 7.4 to allow for two different attack states, one for legato, the other for staccato playing. The rest and skipped rest exit states are accordingly linked to the staccato/legato entry states of the next note. The rationale is that we expect different behaviour of the features if there is a short pause between two notes or not: In the former case, we expect a rise in energy in the attack, in the latter case not, only the PSM should change.

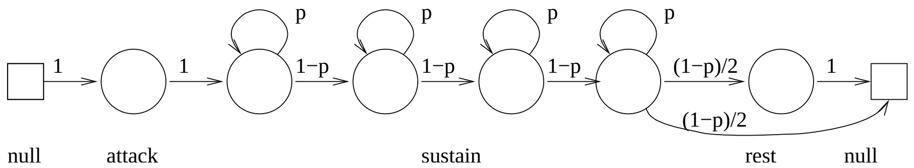


Figure 7.3: Low-level states with linear note model and transition probabilities

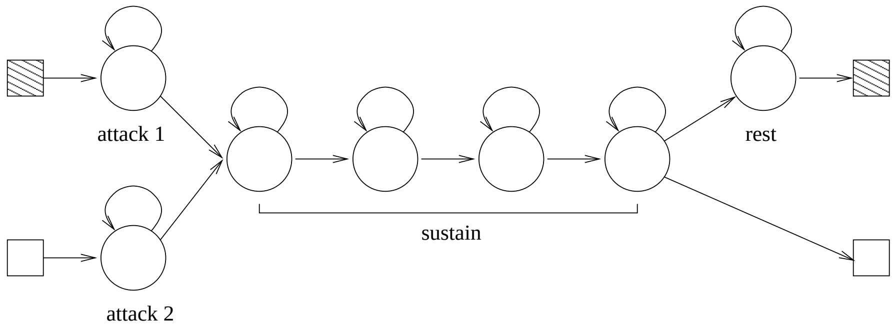


Figure 7.4: Low-level states with two-way note model

7.3.3 Score Model

For each event in the score (as defined in section 5.3), a low level model is created. We call each of these models a *high-level state*. The high level is mostly conceptual and helps us to more easily express a topology for the score modeling. In HMM terms they don't exist because all the models of the high-level states are concatenated to one big model, the entry/exit states being merged to "glue" states between them. Accordingly, there is only one big transition probability matrix, where each note model occupies a square along the diagonal. The transitions between high-level states wind up in the rows of the pseudo "glue" states.

Together with the sequence of events in the score, which have temporal relationships that are reflected in the left-to-right structure of the HMM, also possible performance mismatches are modeled. As introduced by (Dannenberg 1984), there are three possible errors: wrong notes, skipped notes, or inserted notes. The model copes with these errors by introducing error states, or *ghost states*, that model the possibility of a wrong event after each event in the score. Figure 7.5 shows the possible paths for a correct performance (a) and the three possible errors wrong note (b), skipped note (d), where the model has to wait for the next correct note to resynchronise itself to distinguish a skipped note from a wrong note, and extra note (d). Only one ghost state is show for clarity.

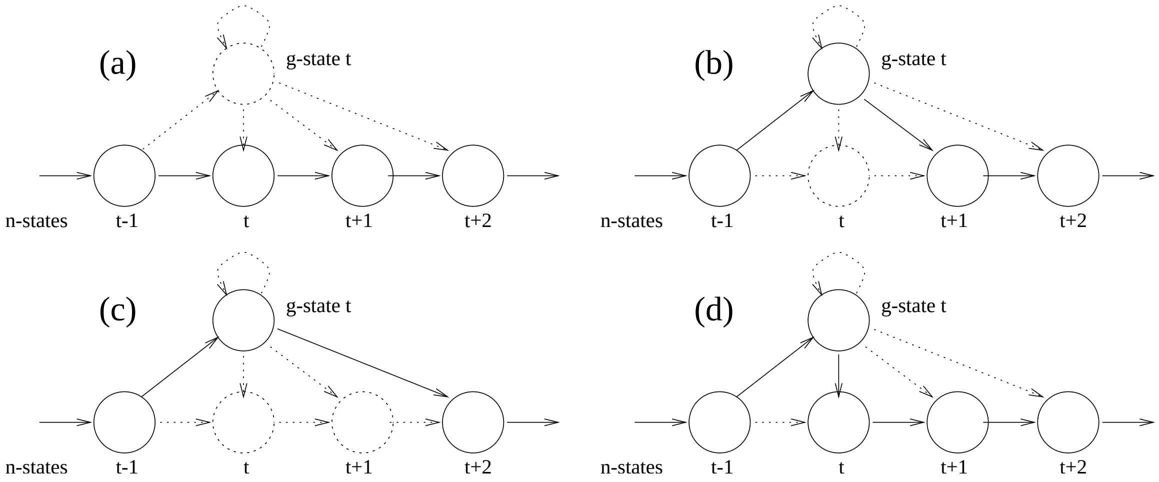


Figure 7.5: High-level states with different possible errors (from Orio and Déchelle 2001)

7.3.4 Decoding

The decoding of the HMM states tells us for each performance frame which state is most probable to be aligned with this frame. The decoding had to be modified for use with real-time score following. The standard Viterbi algorithm can't be used, because we can't wait until the end of the performance to know when the action events should have been triggered during the concert...

So we need another criterion for *real-time decoding*: According to Orio and Déchelle (2001), for each performance frame at time t, we need to perform the following maximisation

$$q(t) = \underset{v(t) \in S}{\operatorname{argmax}} P(q(t) = v(t), o(1) \ldots o(t) \mid \lambda) \tag{7.8}$$

which corresponds to the maximum state index i of the forward variable $\alpha_i(t)$ from equation (7.2).

7.4 Training

Training means adapting the various probabilities and probability distributions governing the HMM to one or more example performances in order to optimise the quality of the alignment. Two different things can be trained: the transition probabilities between the states of the Markov chain, and the probability density functions (PDFs) of the observation likelihoods. While the former needs much example data, especially with errors, the latter can be done by a statistical analysis of the features to derive the PDFs, which essentially perform a mapping from a feature to a probability of attack or sustain or rest.

The model uses thresholded exponential PDFs to calculate the observation likelihoods, because they are better adapted to the acoustic features. The PDF parameters were determined from analysing different sounds, in particular trumpet, clarinet, saxophone, and flute, of the *Studio On Line* (Wöhrmann and Ballet 1999) sound database in MATLAB. Reference alignments for a few audio files were prepared by hand, using a pitch tracker for support that outputs note position and pitch.

The "training" of observation probabilities is done by off-line analysis in MATLAB of feature data written to SDIF files (see section 13.4.1). In order to assign the frames to the low-level state classes they belong to, we have to identify them from the reference alignment and the score. In (Schwarz and Orio 2002), we defined a low-level state class hierarchy depicted in figure 7.6. The statistics are made for the leaf classes, and are then combined upwards in the parent classes. This has two advantages: First, we are more flexible in the study of the statistics, being able to visualise easily a specific class, its parents, up to the whole file. Second, there might be classes which occur rarely, so that not a sufficient number of frames exist to produce a viable statistical estimation of the probability distribution. In that case, we'd use the statistics of the parent class.

The two principal partitions of the state tree are the `note` and the `rest`. The `rest` is simply divided in attack, sustain and release classes. The note sub-classes and their sub-trees are:

Attack states `a`
: `a_detached` after a rest, `a_legato` after a note, `a_same` after the same note

Sustain states `s`
: `s_normal` normal note sustain, `s_trill` note is a trill, `s_fricative` note is a fricative (for the singing voice)

Release states `r`
: The release should distinguish the length of the gap until the next attack, so we'd have: `r_detached`, `r_legato`, `r_same-note`

In the future, when more training data is available, and if we have reliable dynamics information in the note velocity, we can introduce dynamics-dependent sub-states under `s_normal` and `r_detached` to capture the different expected delta-energy profiles in the statistics: `s_forte` and `r_forte` (note is *f*...*fff*), `s_piano` and `r_piano` (note is *p*...*ppp*)

The histogram of values for each class for each feature is computed over one or more soundfiles. See figure 7.7 showing the histograms and statistical parameters (average, standard deviation, threshold, and μ) of an example sound file with singing voice of the features log-energy (len), delta log-energy (dle), peak structure match (psm), delta peak structure match (dpsm), cepstral difference (cepd), and zero crossing rate (zcr), for the state classes attack (a), sustain (s), release (r), note, and rest. The threshold and mean of the exponential PDF is determined by looking for the maximum frequence in the histogram. For delta features, the threshold is based on the standard deviation.

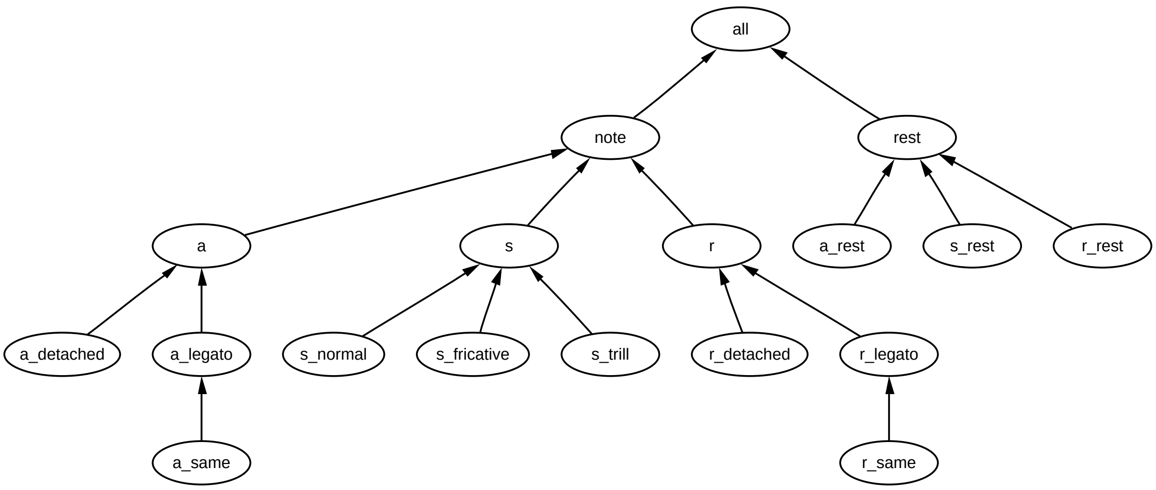


Figure 7.6: Low-level state class tree

The automatic determination of PDF parameters is still experimental. There is no known literature on it, as we know of, because usual HMMs use Gaussian distributions. We must robustly capture the human way of determining the parameters by injecting some prior knowledge about the features.

Then of course a real iterative training (supervised by providing a reference alignment or unsupervised starting from the already good alignment to date) of the transition and observation probabilities is necessary to increase the robustness of the follower even more. This training could adapt to the "style" of a certain singer or musician.

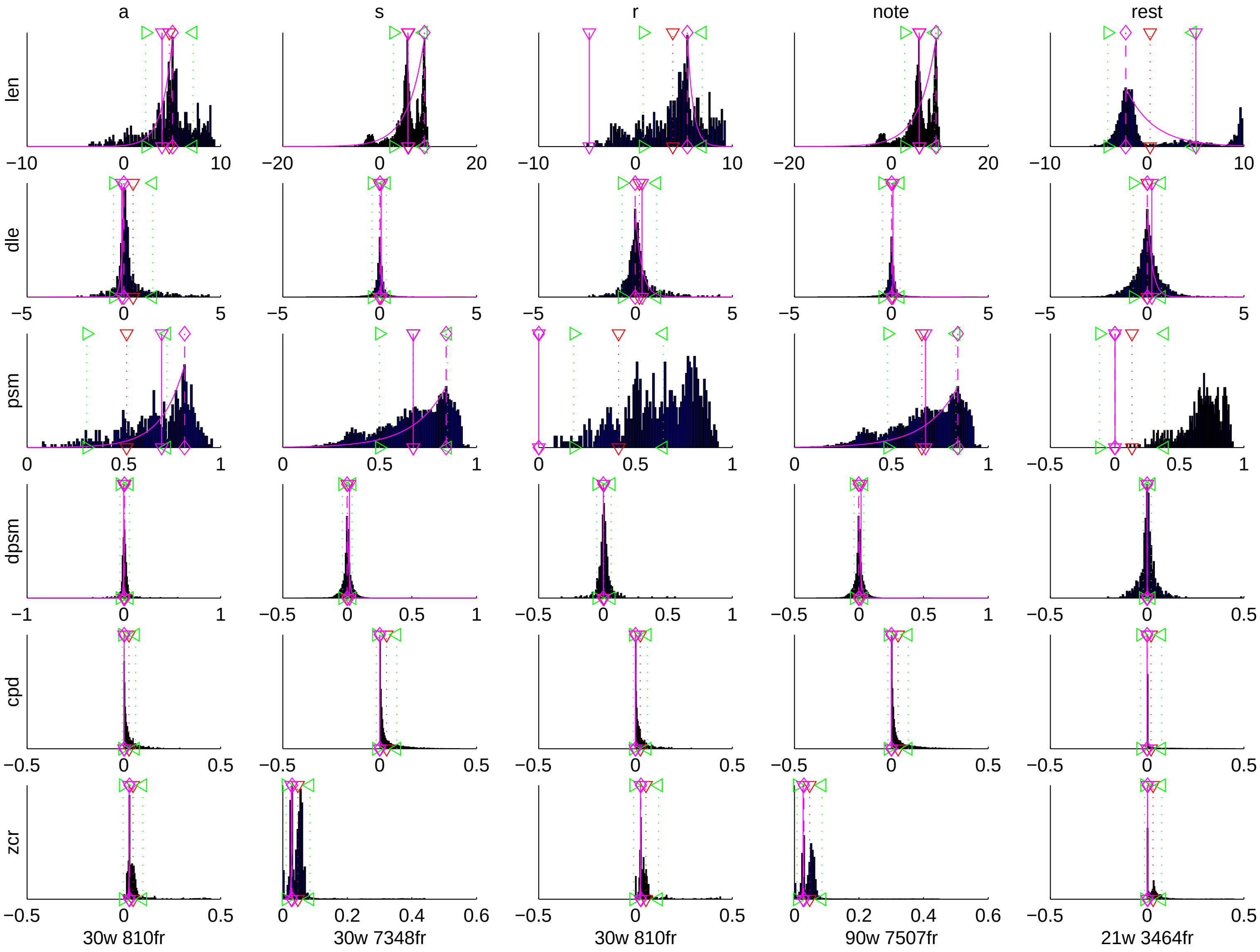


Figure 7.7: Feature histograms of the beginning of section *Riviere* of *En Echo*

7.5 Results of HMM Alignment

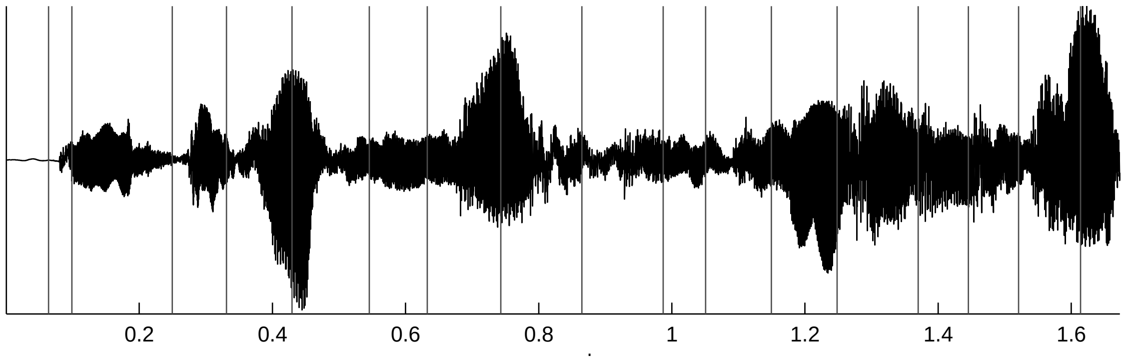
The HMM alignment system is implemented in the real-time interactive environment *jMax* (Déchelle, Cecco, Maggi, and Schnell 1999; Déchelle, Schnell, Borghesi, and Orio 2000).

We performed a subjective evaluaton of alignment by HMM with the "click" method (section 5.6.1) and additional visual feedback: the recognised score event is highlighted in the score display of *jMax*. Even on difficult performances like the very fast intro of *Anthèmes 2* by Boulez, trills, polyphonic performances, and presence of the accompanying orchestra in background in a live concert, the alignment is very precise, triggering the notes with no noticeable delay.

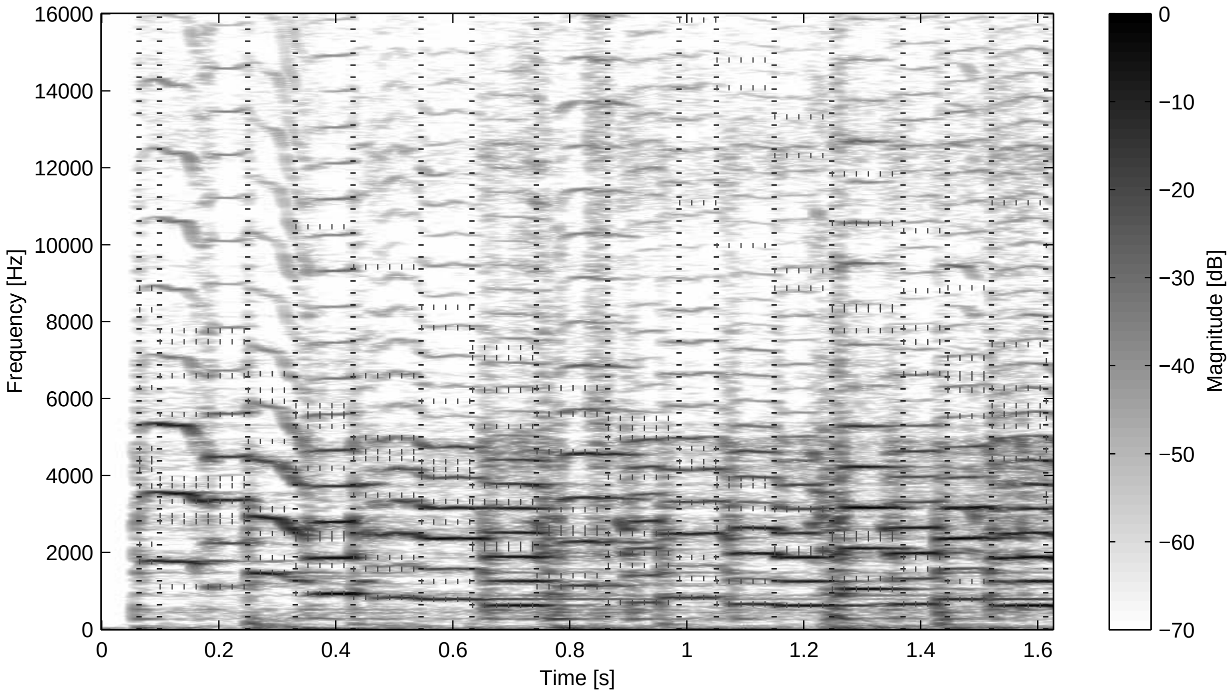
The alignment by HMM has not been applied to the whole corpus of synthesised performances because running hundreds of sound files in batch is not feasible in the interactive *jMax* system.

The figures 7.8 and 7.9 show the results on the wave and the spectrogram of the aligned score (written to SDIF in segment and Midi description types, see appendix E), using the *jMax*–SDIF interface (see section 13.4.3). Although there is some delay visible in the figures, the accuracy is sufficient on monophonic and slightly polyphonic performances. The Mozart quartet, however, could not be aligned, since our filters are much too large, and the real-time requirement means we can not find an acceptable path afterwards. This means, the follower gets stuck.

However, an objective comparison of the HMM alignment results with those obtained by DTW and a discussion can be found in chapter 8.

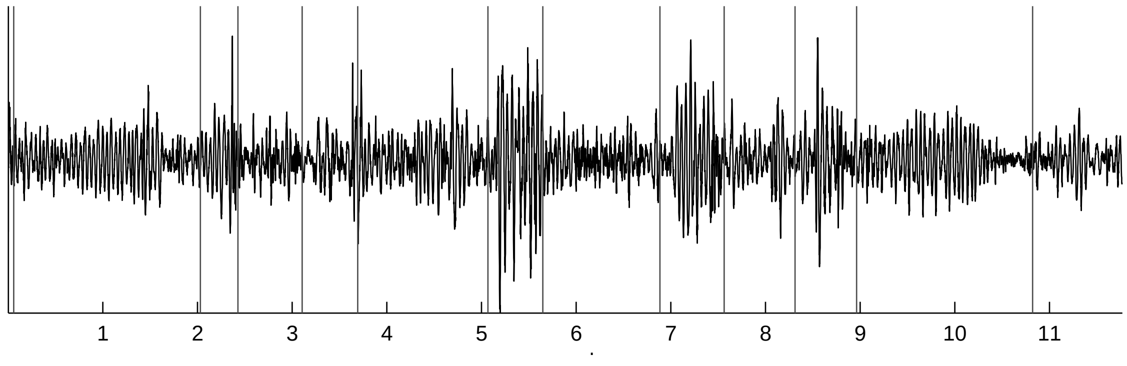


(a) Waveform and alignment marks

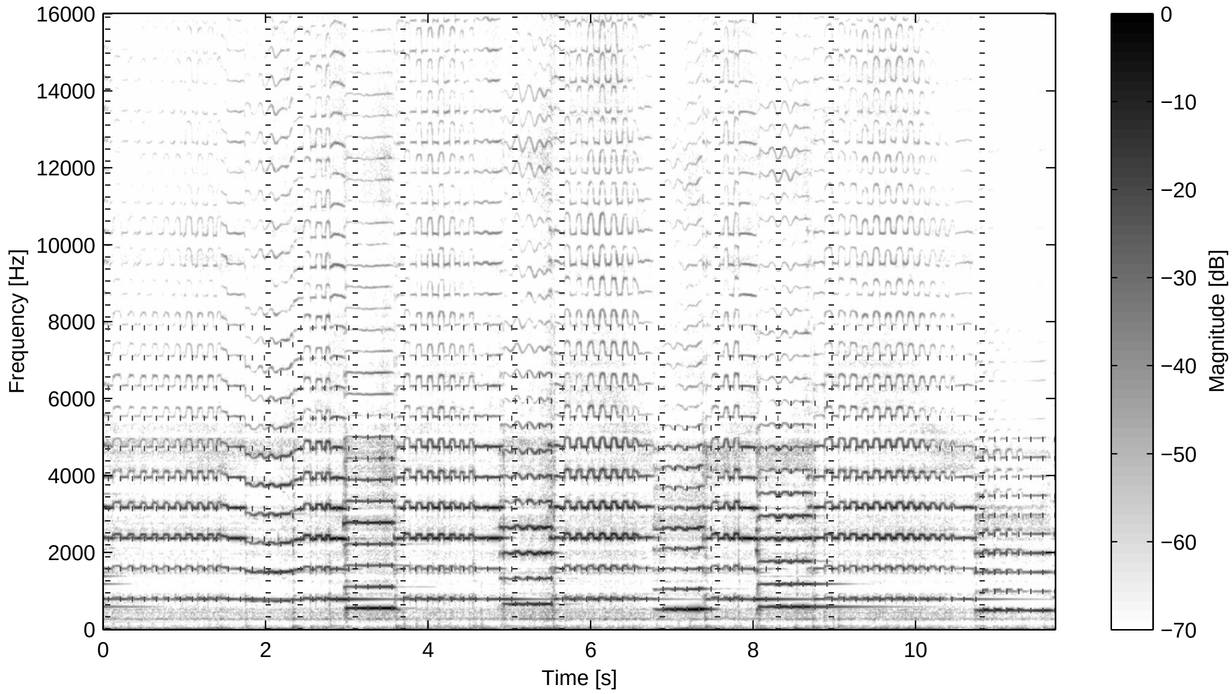


(b) Spectrogram and alignment marks

Figure 7.8: HMM Alignment result for a fast excerpt of the introduction of *Anthèmes 2* by Boulez.



(a) Waveform and alignment marks



(b) Spectrogram and alignment marks

Figure 7.9: HMM Alignment result for an excerpt with trill of *Anthèmes 2* by Boulez.

Chapter 8

Discussion

This chapter summarises and compares the two presented approaches to score–performance alignment by Dynamic Time Warping (chapter 6) and Hidden Markov Models (chapter 7).

8.1 Comparison of DTW and HMM

At the signal level, for both methods the main feature for alignment is the peak structure match (section 5.5), plus energy and deltas. Our DTW method takes full advantage of the non-real-time requirements, and performs a costly but precise spectral matching, and models the attack of a note (sections 6.1.1 and 6.1.2). The HMM method, however, can use more features, since their associated parameters can be trained.

It can be seen that the alignment by DTW and by HMM share many aspects, certainly at the note modeling level: both distinguish the attack, sustain and release phases of a note. However, the advanced two-attack note model for HMM in figure 5.4 can cope with more variation in the performance, and specific adaptation to an instrument, a performance, or a performer is possible by analysing for the observation probabilities, and training the transitions.

At the higher level of the score model, HMMs prove to be more flexible, allowing easy modeling of parallel paths for the score, which is used by means of the ghost states to gracefully handle performance errors. Even more complex configurations are possible: One could model a score where certain musical phrases can be skipped, or repeated an indefinite number of times. Even improvised parts with all notes possible in parallel, can be modeled.

The comparison of the results is for the moment mainly qualitative (section 5.6.1), because of the lack of reference alignment databases—see section 5.6.4 for an initiative to change this unsatisfactory situation. We can still compare the results of DTW and HMM alignment, without knowing which one produced the "better" alignment.

8.2 Conclusion

In general, because of the complete knowledge of the score and performance, DTW is more robust and is more advanced for highly polyphonic and multi-instrument performances.

Name	**num. notes**	**average offset**	**average abs. offset**	**standard deviation of difference**
intro1	16	51.9	57.0	35.3
intro2	7	-53.1	53.1	21.0
tres2	12	-29.0	36.8	45.4

Table 8.1: Comparison of DTW and HMM alignment

Another consequence of the different horizons of DTW and HMM affects the reaction to errors (either performer errors or wrong scores, which is the same from the point of view of the system): DTW using neighbourhood types III and V can not stay on one score frame, such that missing or extra notes will forcibly result in misalignment of some neighbouring notes. The missing notes are accommodated by somehow squeezing them into the path, with some side-effects, because the global path must be monotonic (no zig-zagging possible).

Real-time HMM has a horizon of a window in the score limited to 20 events. Because decoding is in real-time, no alignment path continuity is stipulated, allowing the path to jump forward or backward (zigzag) at any time. This means that the system can react more quickly to errors, e.g. jumping immediately ahead to the next matching note, or can wait on the last recognised note until a new known one arrives. This means also that the system might make a number of wrong recognitions when ambiguous input arrives, jumping back and forth before resynchronising, or even, for larger stretches of errors, that the system can get lost when the next correct note is outside of the event horizon.

For our application of database building, HMM alignment is not accurate enough, because of the real-time requirements, but it performs well for score following, with a small and constant memory required for the event horizon, even for arbitrarily long performances. DTW alignment is precise, but runs 2 hours for 5 minutes of music, occupying 400 MB of memory, despite all the optimisations described in section 6.4.

8.3 Remaining Problems

One fundamental problem restricts the choice of features for alignment: It is difficult to generate good expected values from the score that match the feature values of the performance. This is due to problems of scaling and normalization, and, more difficultly, to the need of a model of the instrument and the performer, see (Dannenberg and Derenyi 1998).

The accuracy of the alignment is limited by the hopsize and suffers from an uncertainty within the window. For staccato performances, we can significantly improve the offset by reanalysing the found frame with a simple but precise energy-based onset detector. For other performances, the accuracy will not be affected.

The integration of percussion alignment is currently being worked on and will allow to align pop music tunes to a Midi score.

Finally, the biggest piece of 7 minutes of music took 900 GB of memory, which is just manageable with today's computers, but a 10 min song is unfeasible since it would need 2 GB of memory. To remedy this, we need a way to split up a performance *and* its score into chunks at the right places, and align them individually. We could also write the (shortcut) path (see section 6.4.2) predecessor matrix ψ to disk line by line and re-read it in inverse order when decoding the best path.

Part III

The Data-Driven Sound Synthesis System CATERPILLAR

"Who are YOU?" said the Caterpillar.
Alice replied, rather shyly, "I–I hardly know, Sir,
just at present — at least I know who I WAS
when I got up this morning, but I think I must
have been changed several times since then."

Lewis Carrol

Chapter 9

System Overview

This Part describes the CATERPILLAR system for data-driven concatenative sound synthesis based on unit selection. The first chapter gives an overview over the system; detailed descriptions of its elements are given in the following chapters. We are here at the pivoting point between Parts II and III, linking the sound databases obtained by automatic alignment with further analysis, storage and selection for synthesis.

The CATERPILLAR software system, described in (Schwarz 2000; Schwarz 2003c; Schwarz 2003a), has been developed to perform data-driven concatenative unit selection sound synthesis. Its elements are illustrated in figure 9.1 and described briefly in the following.

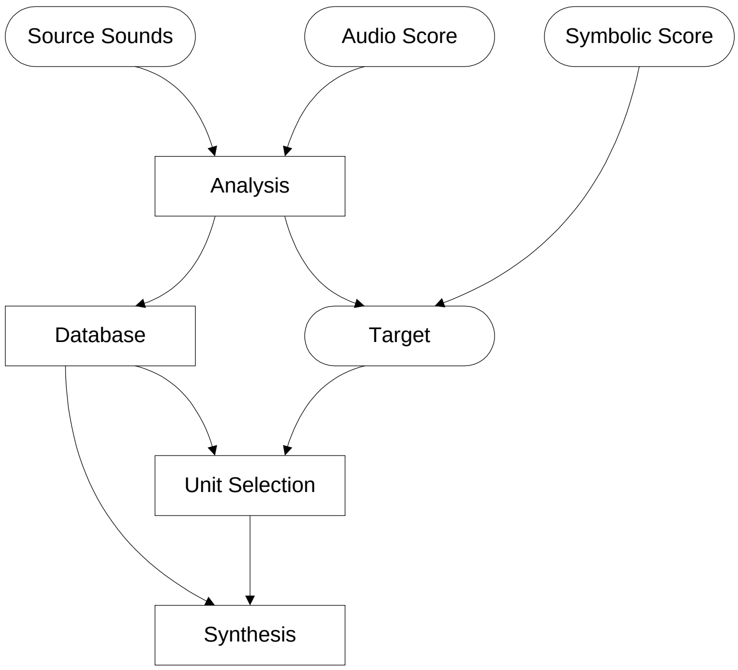


Figure 9.1: Overall structure of the CATERPILLAR system, rounded boxes representing data, rectangular boxes components, and arrows flow of data.

Analysis
: The source sound files are segmented into units (Part II) and analysed to express their characteristics with sound descriptors (10).

Database
: Source file references, units and descriptors are stored in a relational database (12).

Target
The target specification (9.1) is generated from a symbolic score (expressed in notes or descriptors), or analysed from an audio score (using the same segmentation and analysis methods as for the source sounds).

Selection
Units are selected from the database according to given target descriptors and an acoustic distance function (16).

Synthesis
is done by concatenation of selected units with a short cross-fade, possibly applying transformations (16.5).

9.1 Target Specification

The target features can come from two different sources, or be a combination from both:

A *symbolic score*, e.g. from a MIDI file, contains discrete note values and other control parameters, such as volume or brilliance. Also, high-level performance instructions, e.g. legato or staccato, and the lyrics sung for a voice piece can be attached to the symbolic score.

An *audio score* is a recording that is analysed to obtain certain target features. Taking the complete target features from the audio score allows resynthesising it with sounds from the database.

To be usable by the unit selection algorithm, the target specification has to be segmented into a sequence of target units t^τ and the target unit features t^τ_f have to be generated. A symbolic score is segmented by the notes it contains, and the target features are generated from the symbolic information. An audio score is segmented and analysed just like sounds for the database (see Part II and chapters 10 and 11). Depending on the type of synthesis application, the resulting sequence of target units in either case is sub-segmented into the same sub-units as the database units that are to be used for selection. See section 17.1.1 for the sub-segmentation into seminotes, and attack, sustain, and release units.

Chapter 10

Sound Descriptors

A *descriptor* is a value that describes a certain quality of a sound, which can evolve over time or be constant. In speech analysis/synthesis, the usual term is *feature*, as in *feature vector*. Here, the more precise term *descriptor* will be used. The various European and international projects on audio indexing and retrieval described in chapter 2.2 also use this term.

In Caterpillar, we distinguish three levels and three types of descriptors: The low-level descriptors (LLDs) are extracted from the source sounds by signal analysis methods. High-level descriptors (HLDs), that bear a musical sense, are usually attributed to units by the user, or are given by the score. Perceptual descriptors occupy an intermediate level, since they are derived from the absolute measures of low-level descriptors, but are transformed to match the human perception.

The three types of descriptors are determined by the values a descriptor can take: *Category descriptors* are boolean and express the membership of a unit to a category or class, and all its base classes in the hierarchy (e.g. *violin* → *strings* → *instrument*). *Static descriptors* are a constant value for a unit (e.g. Midi note number), and *dynamic descriptors* are evolving over the unit (e.g. fundamental frequency).

The descriptor data types can be boolean, real, int, or symbolic. All this can eventually be represented by a real value in the database, for the sake of canonicity.

Sections 10.3 through 10.7 explain the descriptors used in Caterpillar and how they are calculated. An overview of the descriptor groups in the database is given in figure 10.1. They follow in part the definitions from Rodet and Tisserand (2001) and Lambert (2001a). The following chapter 11 describes the modeling of the temporal evolution of a dynamic descriptor over a synthesis unit.

The following formulas all use the base variables listed in table 10.1, which refer to one window of the signal, which is subsequently analysed by a Fourier transformation (FFT) and a decomposition into sinusoidal harmonic partials (Rodet 1997b).

f_s	sampling rate
N	number of samples in signal window
s_i	digital sound signal samples, $1 \leq i \leq N$
M	number of bins in the first half of the FFT magnitude spectrum ($M = N/2$)
a_i	FFT magnitude of bin i, $1 \leq i \leq M$
f_i	frequency of FFT bin i, given by $f_i = \frac{i}{M}\frac{f_s}{2}$
H	number of harmonic partials
A_i	amplitude of harmonic partial i, $1 \leq i \leq H$
F_i	frequency of harmonic partial i, $1 \leq i \leq H$
F_0	fundamental frequency
K	number of Bark bands in inner-ear filter ($K = 24$)
L_i	specific loudness in Bark band i

Table 10.1: Basic data variables for descriptor calculation

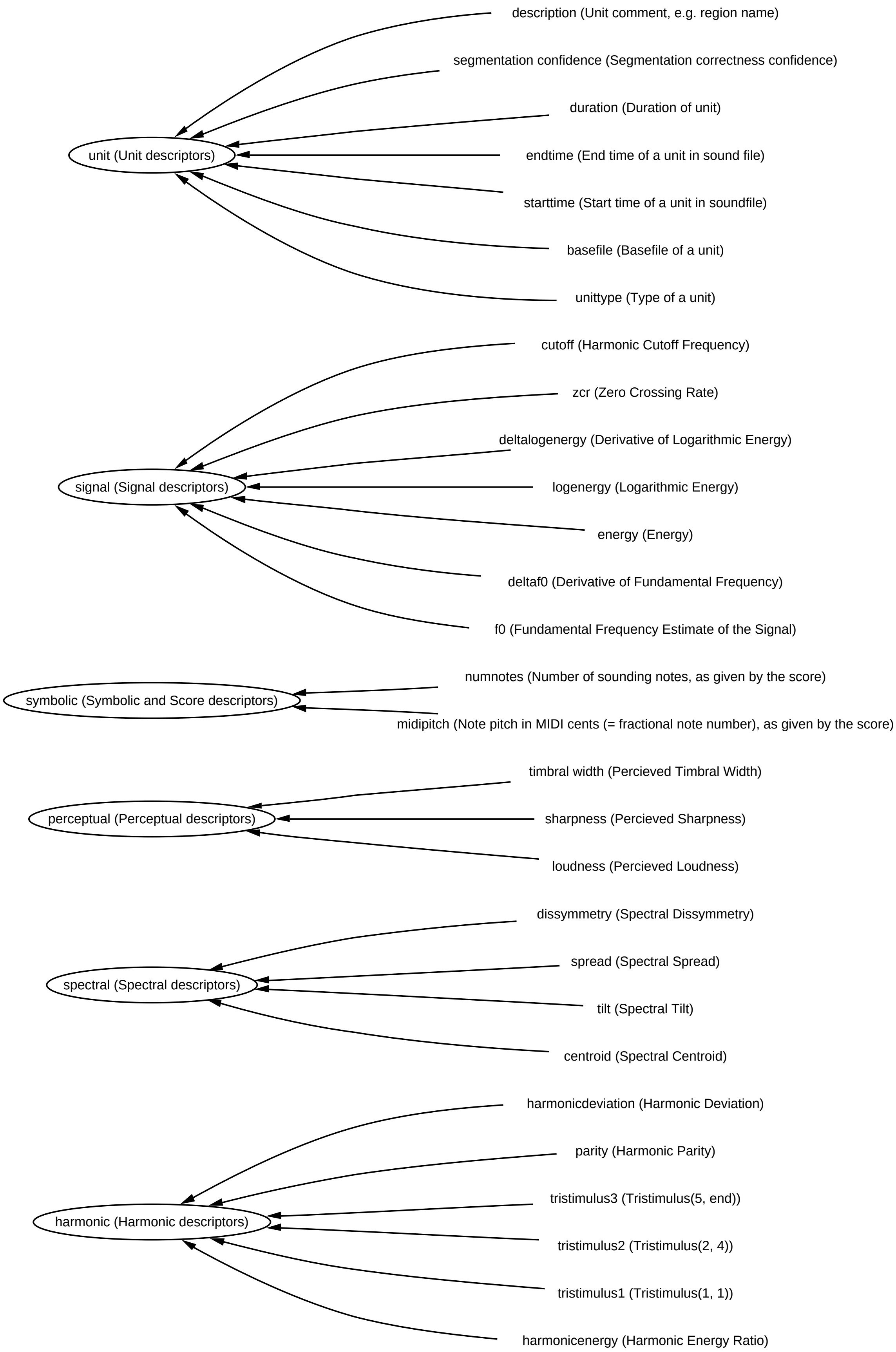


Figure 10.1: Descriptor groups.

10.1 Unit Descriptors

The first group of descriptors, shown in figure 10.2, describe the units themselves. The data is also available in the table Unit (A.2.3), but, to be available for selection, has to be duplicated as descriptors. The descriptors are the start and end time of the unit and its duration, the unit type, the soundfile it came from (basefile ID), and the segmentation confidence, which is a measure of how much the segmentation algorithm believed that the segmentation of this unit was correct.

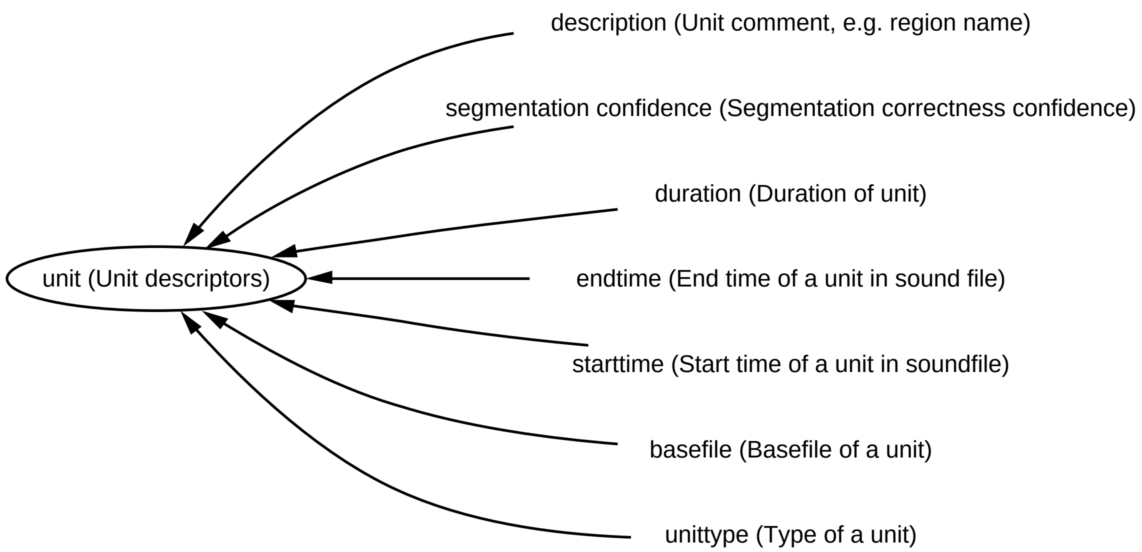


Figure 10.2: Unit descriptors

10.2 Category Descriptors

The principal group of descriptors is the membership of a unit in one or more categories describing the unit under various aspects like sound source, mode of excitation, amplification, etc.

Figure 10.3 shows the whole category hierarchy of the *source* aspect as an overview, followed by the three main branches for instruments, voice, and noises in figures 10.5 through 10.4 for legibility. This hierarchy is by no means complete and is only one of several possible classifications of sound sources[1]. It was grown "on demand" (but with some planning, nevertheless) and will further grow when sounds of new classes will be added to the database.

The categories under the aspect *mode of excitation* shown in figure 10.7 describe, for instruments, the way the sound was produced. The aspect *amplification* shown in figure 10.9 tells us if an instrument produces its sound "acoustically", i.e. by recording with a microphone, or "electrically", i.e. with pickups and possibly amplification effects such as distortion. This aspect serves to distinguish orthogonal sub-classes, like acoustic guitar from electric guitar, violin from electric violin, and acoustic drums from electronic drums.

[1]Nothing prevents us from creating parallel hierarchies of sound source classification, as each unit can be in many classes. Also, the hierarchy does not need to be a tree, i.e. multiple inheritance is possible. The natural semantics for category membership is then that a unit is member of the union of all the base classes of all the classes it is a direct member of.

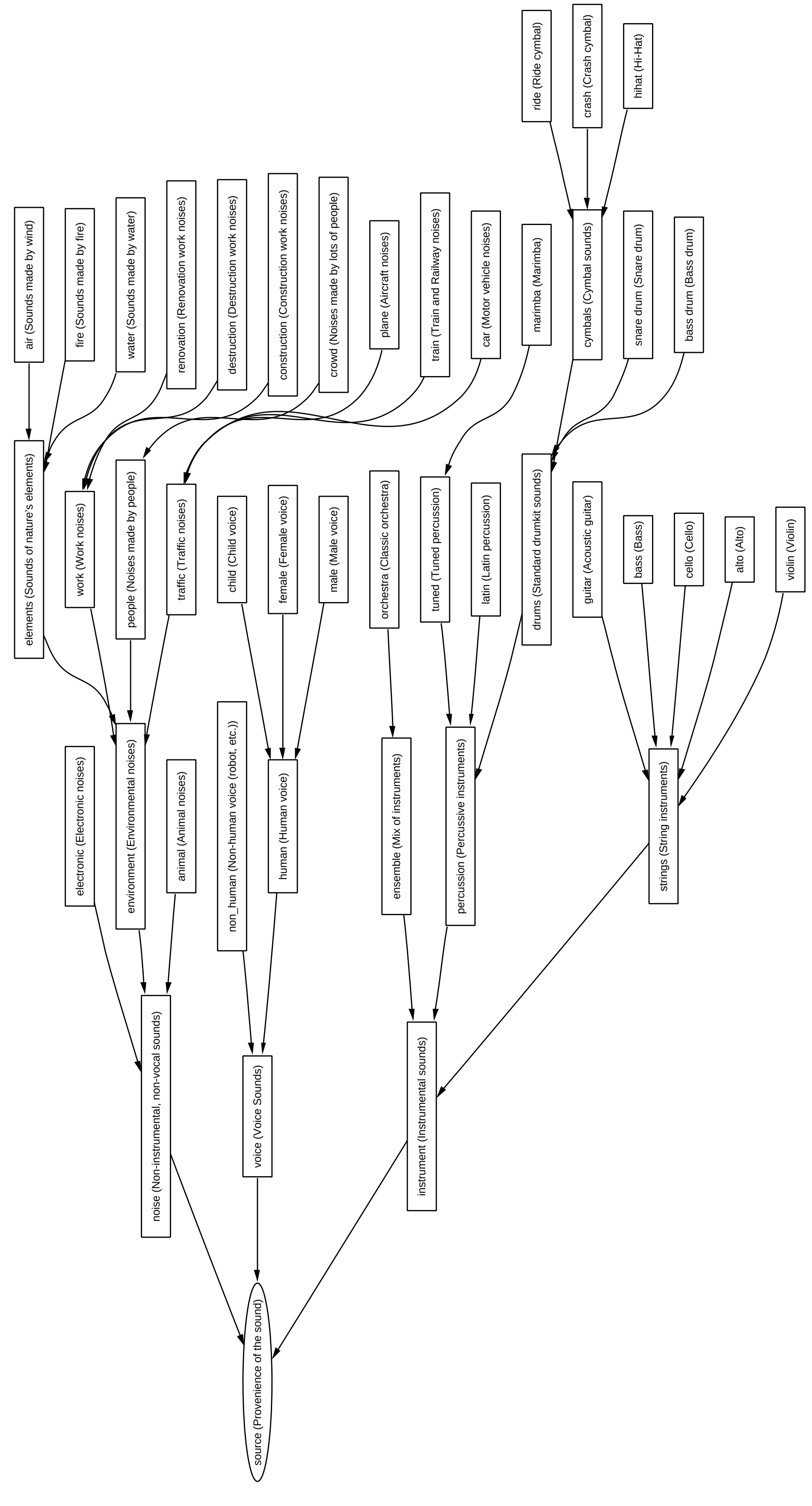


Figure 10.3: Overview of the hierarchy of the source category tree

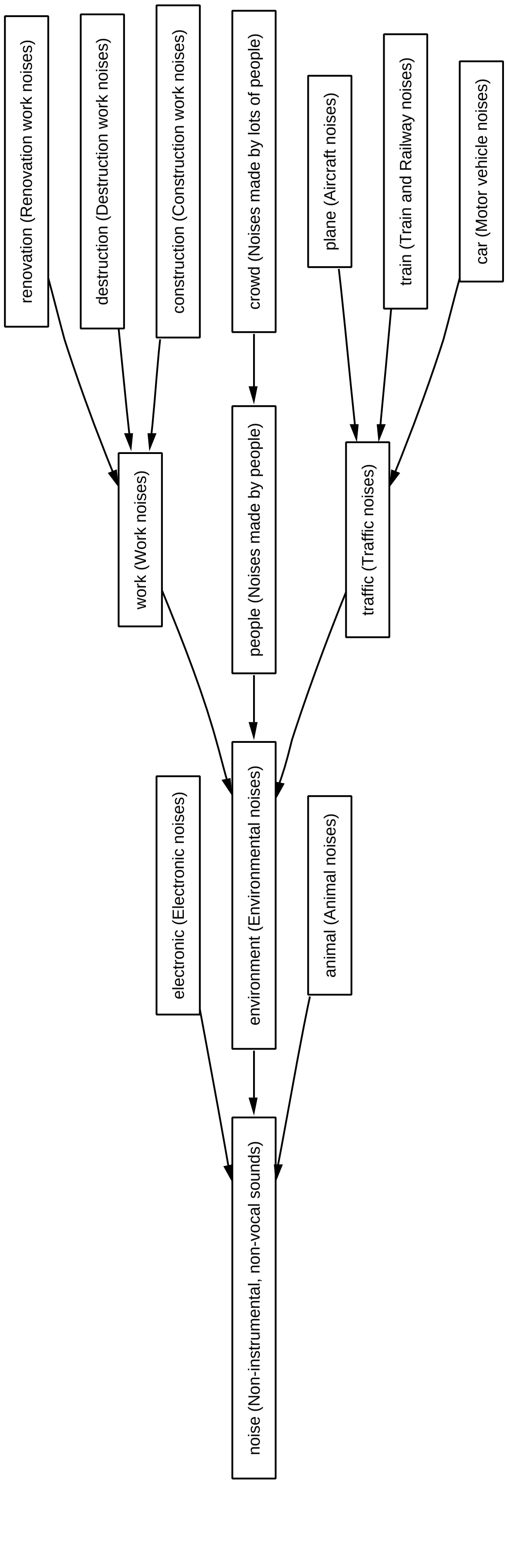


Figure 10.4: Noise sound source hierarchy

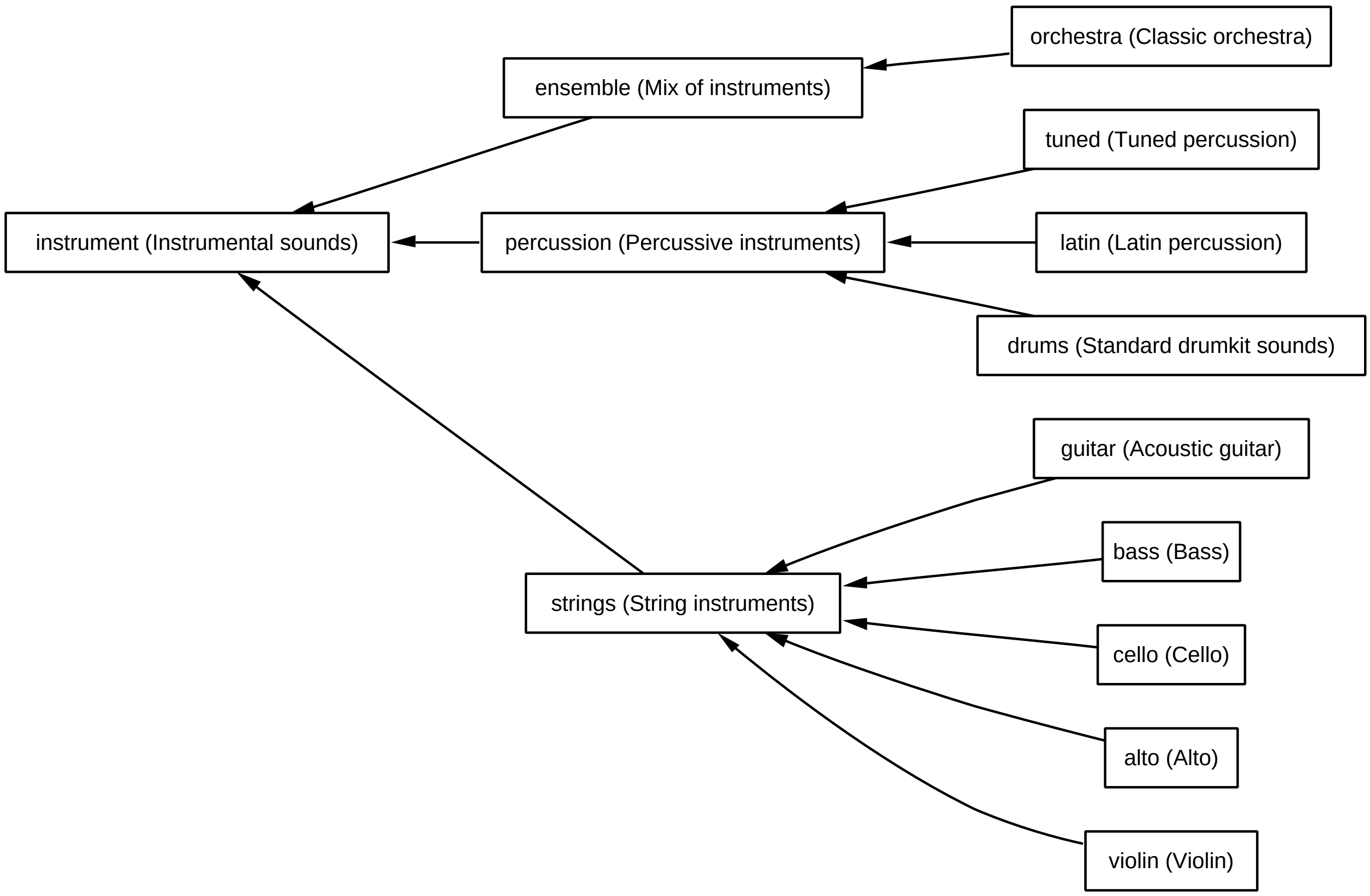


Figure 10.5: Instrument sound source hierarchy

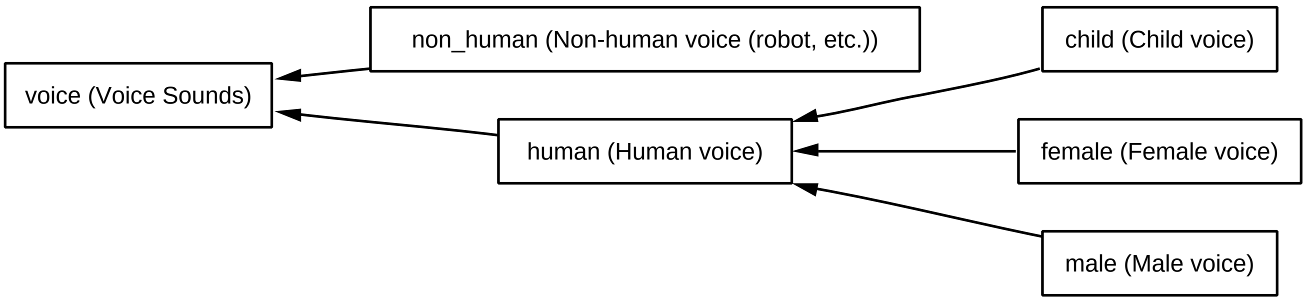


Figure 10.6: Voice sound source hierarchy

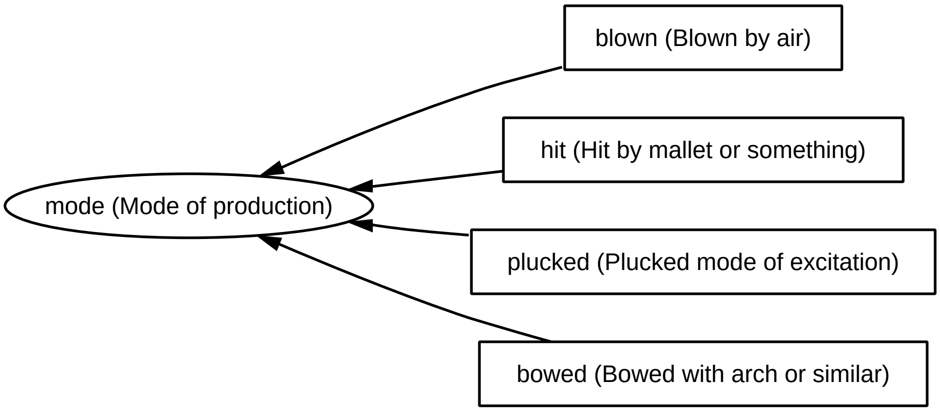


Figure 10.7: Modes of excitation

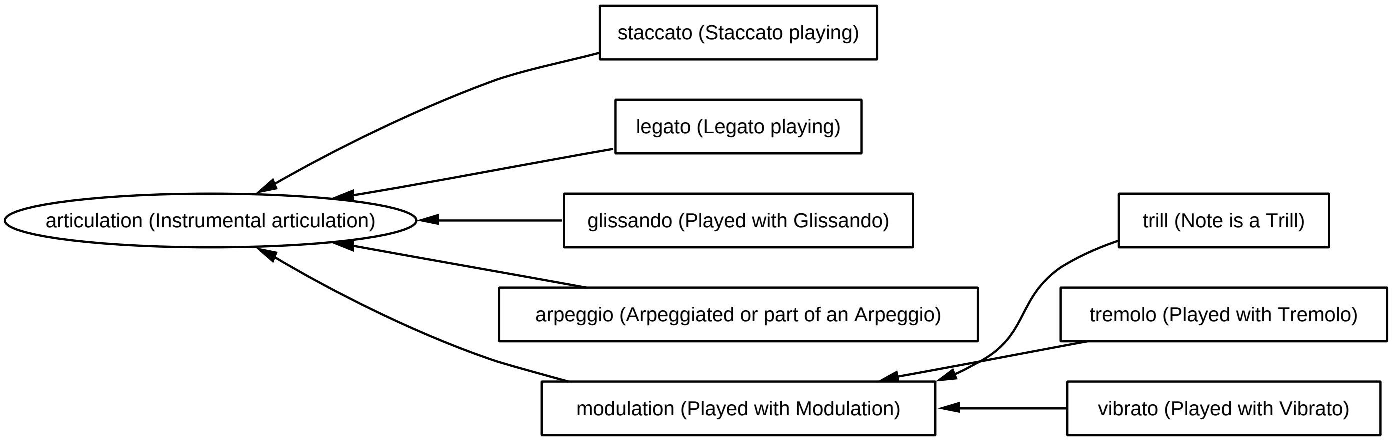


Figure 10.8: Musical articulation categories

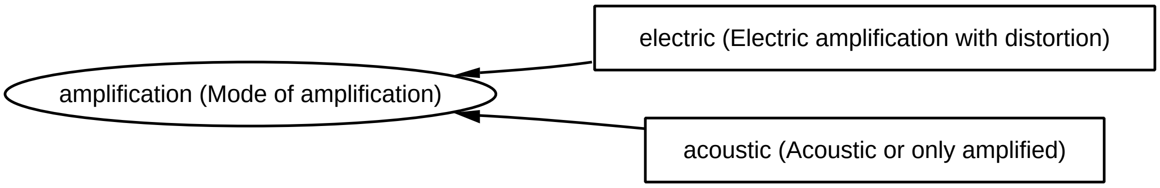


Figure 10.9: Amplification categories

10.3 Signal Descriptors

Signal descriptors (figure 10.10) are directly derived from the sampled sound signal by methods of digital signal processing (DSP).

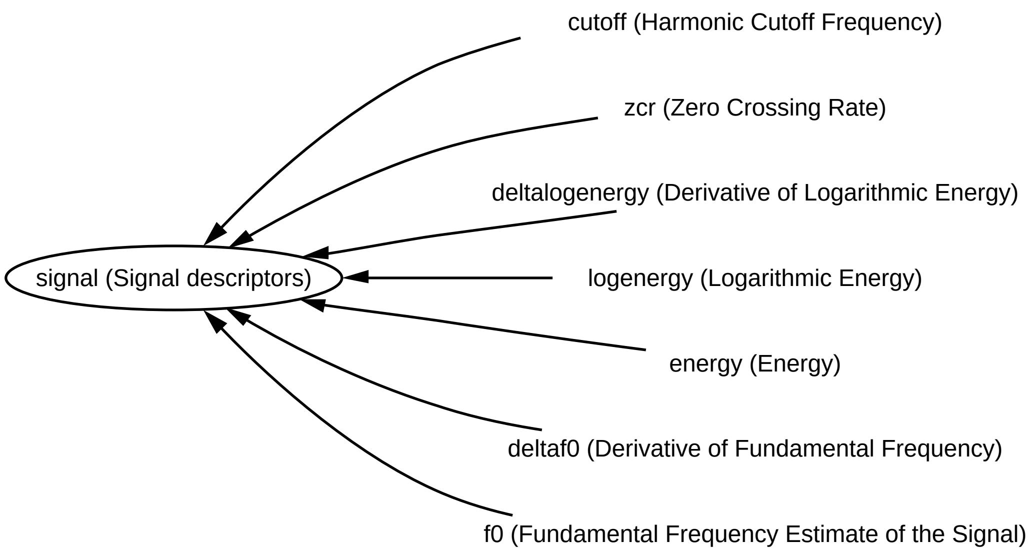


Figure 10.10: Signal descriptors

10.3.1 Energy

The linear energy of the signal is given by

$$Energy = \frac{1}{N} \sum_{i=1}^{N} (s_i)^2 \tag{10.1}$$

10.3.2 Logarithmic Energy

The logarithmic energy is the descriptor *Energy* in decibel (dB), given by

$$LogEnergy = \mathrm{dB}\,(Energy) = 10 \log_{10}(Energy) \tag{10.2}$$

The unit decibel is defined as the ratio R of two signals with energies E_1 and E_2:

$$R = 10 \cdot \log_{10} \frac{E_1}{E_2} \tag{10.3}$$

Often, however, an implicit reference energy of $E_2 = 1$ is used. A ratio of 10 dB corresponds roughly to doubling the loudness. The unit decibel is well suited to practical use, since the dynamic range of the human ear is quite large, still, the values in decibel stay in a manageable range. For example, if the threshold of hearing is defined as a reference point of 0 dB, the dynamic range reaches up to 120 dB, the threshold of pain. This corresponds to doubling the loudness 12 times.

10.3.3 Derivative of Logarithmic Energy

The derivative of logarithmic energy describes more precisely the energy evolution over a unit. It is given, for each analysis frame $i \geq 2$, by

$$\mathit{DeltaLogEnergy}(i) = \mathit{LogEnergy}(i) - \mathit{LogEnergy}(i-1) \tag{10.4}$$

10.3.4 Fundamental Frequency

The fundamental frequency *F0* is the physical correlate of the perception of pitch. It is here calculated by a peak-picking method described in (Rodet 1997b). Various other approaches to the non-trivial problem of pitch detection exists, such as autoregression based on perception (de Cheveigné and Kawahara 2002; de Cheveigné and Henrich 2002; de Cheveigné 2002), probabilistic modeling (Rodet and Doval 1992; Doval 1994; Prudham 2002).

10.3.5 Derivative of Fundamental Frequency

The derivative of fundamental frequency describes the pitch evolution over a unit. For example, it helps to distinguish a portamento continuous pitch change (figure 10.11(a)) from a step-wise glissando or arpeggio (figure 10.11(b)), which can not be done by the *polynomial slope* characteristic value (see section 11.2). It is given, for each analysis frame $i \geq 2$, by

$$\mathit{DeltaF0}(i) = F_0(i) - F_0(i-1) \tag{10.5}$$

10.3.6 Zero Crossing Rate

The zero crossing rate (ZCR) is the number of times the sampled signal waveform crosses the zero level. It gives a very rough indication of the fundamental frequency for harmonic signals, and is higher for noisy parts of the signal. Therefore, it can be used to detect attacks or fricatives. However, the first order autocorrelation (section 10.3.7) is a better alternative for that. The zero crossing rate is computed as:

$$\mathit{ZCR} = \frac{1}{N} \sum_{i=1}^{N-1} sgn(s_i) \textbf{ xor } sgn(s_{i+1}) \tag{10.6}$$

10.3.7 First Order Autocorrelation

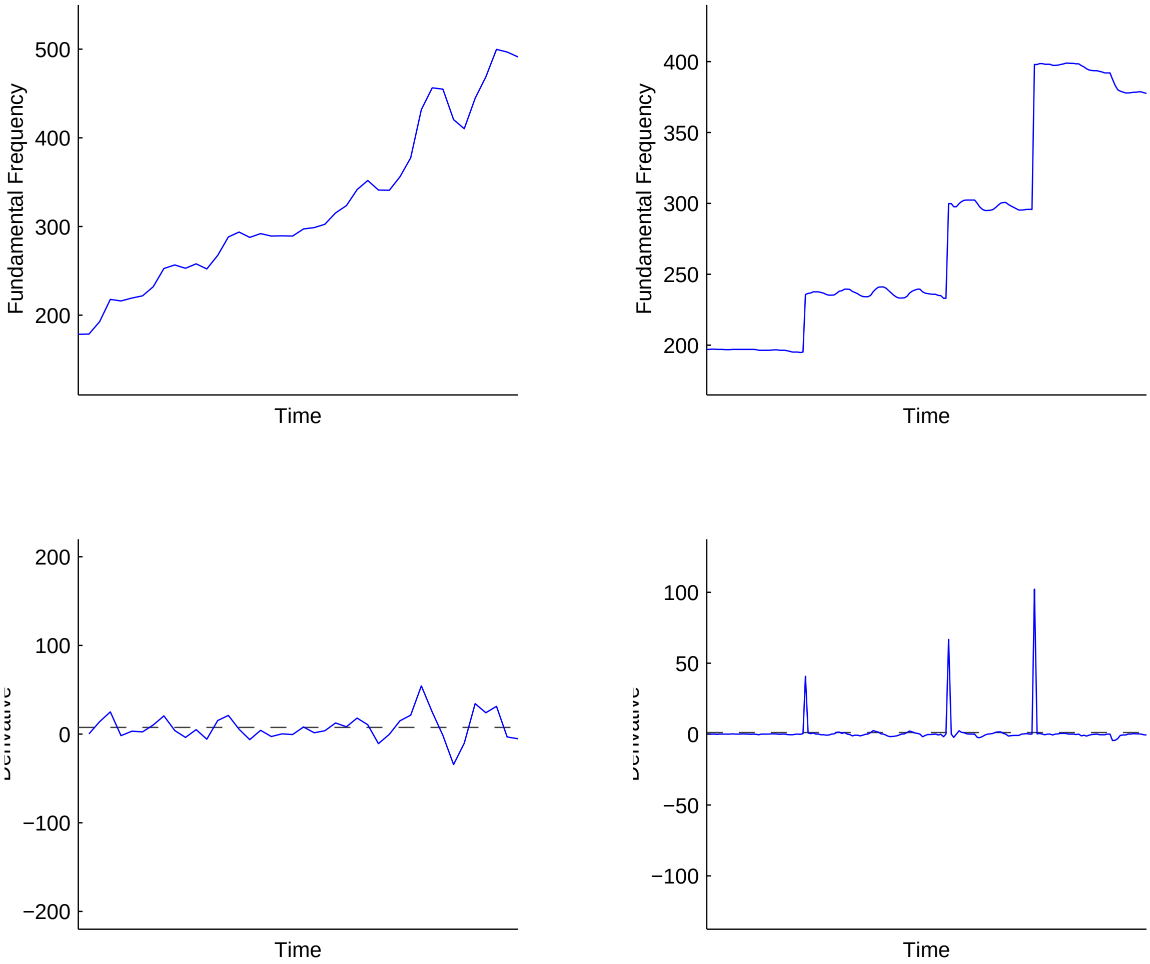
The result from the calculation of the first order autocorrelation coefficient gives an indication of the noisiness of a signal (or non-harmonicity). It can therefore be used to detect attacks or fricatives, with better results than the zero crossing rate (section 10.3.6). It is calculated as:

$$\mathit{AR1} = \frac{1}{N} \sum_{i=1}^{N-1} s_i \cdot s_{i+1} \tag{10.7}$$

10.4 Symbolic and Score Descriptors

Symbolic and score descriptors (figure 10.12) can come from two sources:

- When the units have been segmented by alignment of the source sound file to its score (see Part II), the link between the note units and score can be used to extract the information from the score attached to the notes.



(a) Continuous pitch

(b) Pitch steps

Figure 10.11: Fundamental frequency and derivative of fundamental frequency: continuous portamento vs. step-wise glissando pitch curves.

- Arbitrary symbolic, static or dynamic information can be attached to units, such as subjective judgments like "frightening", or more precise source designations. These are typically expressed as category membership. In the case of continuous subjective descriptors, e.g. a composer would like to rate her sounds on a "glassiness" scale, these can be entered manually (or by file input).

10.4.1 MIDI Pitch

Most often, the score would be a MIDI-file (see section 5.4.5. While this is a very poor representation for a musical score, we can still extract the notated pitch for each note and store it as a constant descriptor. Its unit is *MIDI cents*, a floating point value given by the *MIDI Note Number* as the integer part, plus a fine-tuning in one hundredth half-note steps in the fractional part, to express quarter and eighth tones

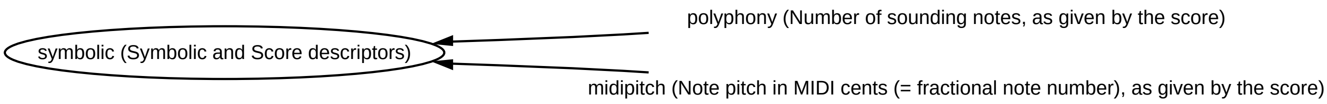


Figure 10.12: Symbolic and score descriptors

10.4.2 Polyphony

For most instruments, the score will be at least partly polyphonic. To distinguish monophonic from polyphonic notes, we store the number of notes in the unit as a static feature.

10.4.3 Lyrics

For vocal sounds, when lyrics are given in the score (as Midi text events), we store the syllable corresponding to the note as ASCII text or in the X-SAMPA phonetic notation (see section F), if given.

10.4.4 Other Score Information

Various other information can be written in the score. It is important to capture this information, because it is situated on a very high musical level, commanding various acoustic effects on the signal level. By performing unit selection according to this high level information, we automatically obtain the desired effects in the sound.

The *playing style* can be given in the score, as playing instructions. For example, *pizzicato* is expressed as the *plucked* category in the *mode of excitation* aspect, whose hierarchy tree is given in figure 10.7. Furthermore, legato or staccato playing can be noted in the score, if the representation supports it. As this is not the case with Midi files, we did not yet define the categorisation aspect *articulation* for it. The same applies to the dynamics *ppp*. . . *fff*, which, in Midi is not represented directly. It could be derived from velocity, but this is not very reliable. Anyway, this playing instruction is not very high-level, because it is very closely correlated with loudness.

10.5 Perceptual Descriptors

The actual perception of many of the physical magnitudes is transformed by the influence of the inner ear. How the physical stimuli are perceived is studied by the science of *psychoacoustics*, dating back to von Helmholtz (1913). A standard introduction to psychoacoustics is Zwicker (1982) or Moore (1989), musical implications are treated in Cook (1999).

10.5.1 Loundness

For the calculation of the perceptual descriptors, we simulate the filtering effect due to the physiological characteristics of the inner ear on the sound (Moore, Glasberg, and Baer 1997). The human auditory system can be modeled by 24 critical bands called *Bark bands*. We designate the specific loudness for a band i by L_i and the total loudness as

$$Loudness = \sum_{i=1}^{K} L_i \tag{10.8}$$

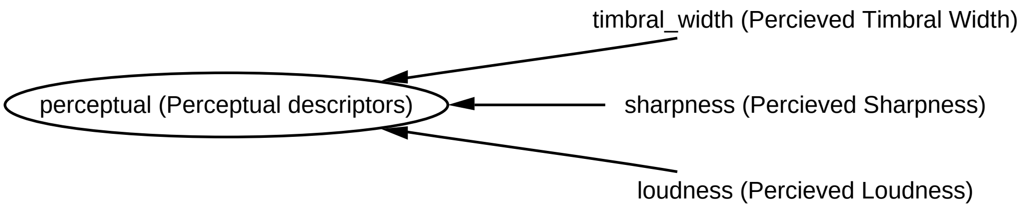


Figure 10.13: Perceptual descriptors

10.5.2 Sharpness

The sharpness expresses the relative high frequency content. It is defined as the average of the specific loudnesses, weighted by the index of the Bark band index and a function

$$g(i) = \begin{cases} 1 & \text{if } \; i < 15 \\ 0.066\, e^{0.171 i} & \text{if } \; i \geq 15 \end{cases} \tag{10.9}$$

by the formula

$$\mathit{Sharpness} = 0.11 \frac{\sum_{i=1}^{K} i\, g(i)\, L_i}{\mathit{Loudness}} \tag{10.10}$$

10.5.3 Timbral Width

The timbral width is the ratio between the maximum specific loudness $L_{\max}$ given by

$$L_{\max} = \max_{1 \leq i \leq K} L_i \tag{10.11}$$

and the total loudness:

$$\mathit{TimbralWidth} = \left(\frac{\mathit{Loudness} - L_{\max}}{\mathit{Loudness}} \right) \tag{10.12}$$

10.6 Spectral Descriptors

Spectral descriptors figure 10.14 describe the shape of the short-time FFT magnitude spectrum by means of statistical and linear modeling.

10.6.1 Spectral Centroid

The spectral centroid (measured in Hertz) is defined as the center of gravity of the FFT magnitude spectrum. It is closely related to the perception of *brightness*.

$$\mathit{SpectralCentroid} = \frac{\sum_{i=1}^{N} a_i f_i}{\sum_{i=1}^{N} a_i} \tag{10.13}$$

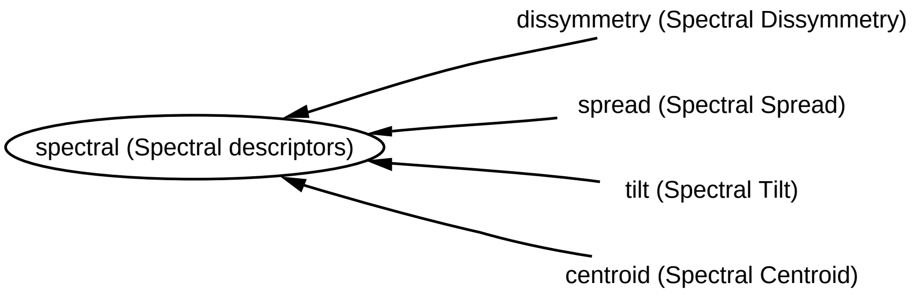


Figure 10.14: Spectral descriptors

10.6.2 Spectral Tilt

Acoustic instruments, such as the trumpet and the voice have source spectra which can vary greatly, especially according to the *physical effort*, i.e. the intensity at which the instrument is played (Beauchamp 1975; Benade 1976; Beauchamp 1980; Bennett and Rodet 1989; Fletcher and Tarnopolsky 1999). The louder the sound, the stronger the high frequency components become. This can be seen as a downward spectrum tilt (or slope) which decreases with loudness. For speech, it is among others responsible for the prosodic feature of *accent*, in that a speaker modifies the tilt (raising the slope) of the spectrum of a vowel, to put stress on a syllable (Dogil 1995).

The spectral tilt descriptor gives an idea of the inclination of the spectrum. Its unit is decibel per octave. For the spoken voice, the typical value is $-6\frac{\mathrm{dB}}{\mathrm{oct}}$

According to Lambert (2001a), we can use the cubic root of the 3$^{\mathrm{rd}}$ order centered moment of the spectral envelope (resulting from additive analysis) as an estimation for the spectral tilt:

$$\sqrt[3]{\frac{\sum_{i=1}^{H} A_i \left(F_i - \mathit{HarmonicSpectralCentroid}\right)^3}{\sum_{i=1}^{H} A_i}} \tag{10.14}$$

with the centroid of the harmonic spectral envelope given by

$$\mathit{HarmonicSpectralCentroid} = \frac{\sum_{i=1}^{H} A_i F_i}{\sum_{i=1}^{H} A_i} \tag{10.15}$$

However, it is preferable to base the spectral descriptors on the FFT magnitude spectrum, and not on the harmonic spectrum, because the latter is not always defined, e.g. for noisy sounds.

We prefer therefore the more robust method suggested by Rodet and Tisserand (2001), where the spectral tilt m is calculated by fitting a regression line $\hat{a}$ to the FFT magnitude spectrum with

$$\hat{a}_i = m f_i + b \tag{10.16}$$

where the square difference $E = \sum_{i=1}^{N} (a_i - \hat{a}_i)^2$ is minimised, yielding

$$m = \frac{N \sum_{i=1}^{N} f_i a_i - \sum_{i=1}^{N} f_i \sum_{i=1}^{N} a_i}{N \sum_{i=1}^{N} f_i^2 - \left(\sum_{i=1}^{N} f_i\right)^2} \tag{10.17}$$

The factor m is per frequency step of an FFT-bin of $f_s/(2M)$ Hz. To converted to the right unit decibel per octave, we have to divide by the number of octaves in the spectrum, $log_2 \frac{f_s}{2} = log_2 f_2 - 1$ (Schwarz 1998; Rodet and Schwarz 2000):

$$\mathit{SpectralTilt} = 10 \log_{10} m \frac{M}{\log_2(f_s) - 1} \tag{10.18}$$

10.6.3 Spectral Spread

The spectral spread is defined as the value of the second order centered moment of the magnitude spectrum (corresponding to the standard deviation):

$$SpectralSpread = \sqrt{\frac{\sum_{i=1}^{N} a_i \left(f_i - SpectralCentroid\right)^2}{\sum_{i=1}^{N} a_i}} \tag{10.19}$$

10.6.4 Spectral Dissymmetry

The spectral dissymmetry is defined as the value of the third order centered moment of the magnitude spectrum (corresponding to the spectral skewness):

$$SpectralDissymmetry = \sqrt[3]{\frac{\sum_{i=1}^{N} a_i \left(f_i - SpectralCentroid\right)^3}{\sum_{i=1}^{N} a_i}} \tag{10.20}$$

10.7 Harmonic Descriptors

The harmonic descriptors in figure 10.15 are calculated from the results of a sinusoidal harmonic analysis, often called additive analysis, where a signal is considered as a sum of H harmonic partials with frequencies F_i and amplitudes A_i, plus a residual noise component. Each frequency F_i is centered around iF_0, the perfect i^{th} harmonic. This means that the harmonic analysis depends heavily on the fundamental frequency analysis.

Additive analysis goes back to Risset and Mathews (1969) and Serra and Smith (1990). Current research and systems performing additive harmonic analysis are described in (Rodet 1997a; Pielemeier and Wakefield 1996; Wakefield 1998; Fitz, Haken, and Christensen 2000b; Fitz, Haken, and Christensen 2000a; Serra, Bonada, Herrera, and Loureiro 1997). See section 3.1.2.3 for the use of additive analysis/synthesis in speech, there called *Harmonics plus Noise Model* (HNM).

A comparative evaluation of different methods and systems for additive analysis–synthesis was the topic of a panel session at the International Computer Music Conference (ICMC) 2000[2]. It was organised and moderated by Matthew Wright. The participants were James Beauchamp (University of Illinois at Urbana-Champaign), Xavier Serra and Maarten de Boer (Music Technology Group, Audiovisual Institute, Pompeu Fabra University), Axel Roebel (CCRMA, now IRCAM), Kelly Fitz and Lippold Haken (CERL Sound Group), Xavier Rodet and Diemo Schwarz (IRCAM), and Gregory Wakefield (University of Michigan). The exchange of analysis result data necessary for comparison was made much easier by the use of the SDIF standard (section 13.4.1). The results are described in (Wright, Beauchamp, Fitz, Rodet, Röbel, Serra, and Wakefield 2000).

In speech synthesis, the additive method is usually referred to as *Harmonics plus Noise Model* (HNM). It is described in section 3.1.2.3 and in (Stylianou 1996; Stylianou 1998a; Stylianou, Dutoit, and Schroeter 1997; Macon and Clements 1995; Macon and Clements 1996)

Harmonic descriptors are not necessarily applicable to all sounds in the database, because there are sounds that do not have a strong harmonic content, such as noisy sounds. Whether harmonic descriptors are applicable to a sound or not is expressed by the *HarmonicEnergyRatio* described below.

10.7.1 Harmonic Energy Ratio

The harmonic energy ratio expresses the amount of signal energy resulting from harmonic partials over the total energy of the signal:

$$HarmonicEnergyRatio = \frac{\sum_{i=1}^{H} A_i^2}{Energy} \tag{10.21}$$

[2] `http://cnmat.cnmat.berkeley.edu/SDIF/ICMC2000/`

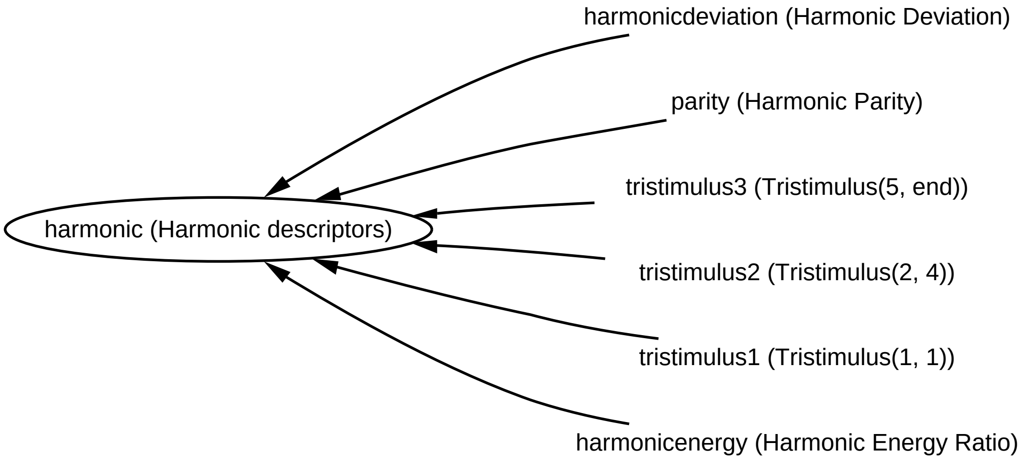


Figure 10.15: Harmonic descriptors

10.7.2 Harmonic Parity

The harmonic parity is the ratio between the sum of the amplitudes of the even harmonic partials and the sum of the amplitudes of all the harmonic partials:

$$\mathit{HarmonicParity} = \frac{\sum_{i=1}^{H/2} A_{2i}}{\sum_{i=1}^{H} A_i} \tag{10.22}$$

It should normally be close to 1, but for certain instruments like the clarinet, where only even partials are present.

10.7.3 Tristimulus

The tristimulus is the sum of the amplitudes of harmonic partials in an index range $i_s \dots i_e$ over the sum of the amplitudes of all the harmonic partials (Jehan 1997).

$$T(i_s, i_e) = \frac{\sum_{i=i_s}^{i_e} A_i}{\sum_{i=1}^{H} A_i} \tag{10.23}$$

We use three tristimuli with the following index ranges:

$$\mathit{Tristimulus1} = T(1, 1) \tag{10.24}$$
$$\mathit{Tristimulus2} = T(2, 4) \tag{10.25}$$
$$\mathit{Tristimulus3} = T(5, H) \tag{10.26}$$
$$\tag{10.27}$$

10.7.4 Harmonic Deviation

The harmonic deviation is the difference between the frequency of the i^{th} partial F_i and its ideal harmonic frequency iF_0. To increase the influence of the loud partials, we weight by the partial's amplitude A_i.

$$\mathit{HarmonicDeviation} = \frac{\sum_{i=1}^{H} A_i \frac{F_i - iF_0}{F_0}}{\sum_{i=1}^{H} A_i} \tag{10.28}$$

Chapter 11

Characteristic Values

Contrary to the categorical or static descriptors detailed in the last chapter, a dynamic descriptor of a database unit varies over time, i.e. we obtain a vector of descriptor values, with one element for each analysed signal frame from the sound file. However, in the database, we need to characterise a unit with a fixed number of scalar values to efficiently query and select units. These scalar values are called *characteristic values* because they express various aspects or characteristics of a unit, e.g. its mean value, its general evolution over time, and so on. In short, we perform a mapping from a variable-length feature vector to a fixed-length condensed description as characteristic values.

We can group the characteristic values into those describing the value or range of the descriptor (section 11.1), those describing the temporal evolution (section 11.2), and those describing the spectral characteristics of the evolution of the descriptor (section 11.3). See figure 11.1 for an example of the characteristic values implemented in CATERPILLAR on the *loudness* descriptor of a database unit. For the temporal modeling, we make use of Legendre polynomial approximation, which is explained in section 11.4. Finally, to unify all descriptors, we define the default characteristic values provided for categorical and static descriptors in section 11.5.

11.1 Value Characteristics

The value characteristics describe the descriptor value for the unit. We characterise a unit by:

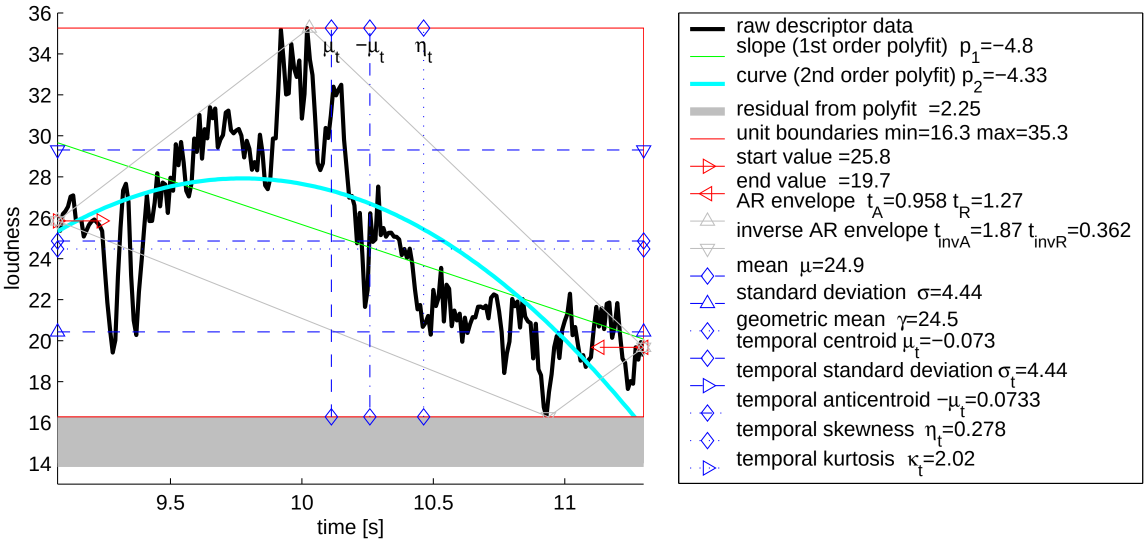
Mean
: The average or arithmetic mean descriptor value is the main feature used for selection.

Geometric Mean
: For fundamental frequency, the geometric mean $\sqrt[n]{\prod_{i=1}^{n} x_i}$ with x_i the n descriptor values, corresponds better to the perceived pitch than the arithmetic mean for notes with vibrato (Shonle and Horan 1980).

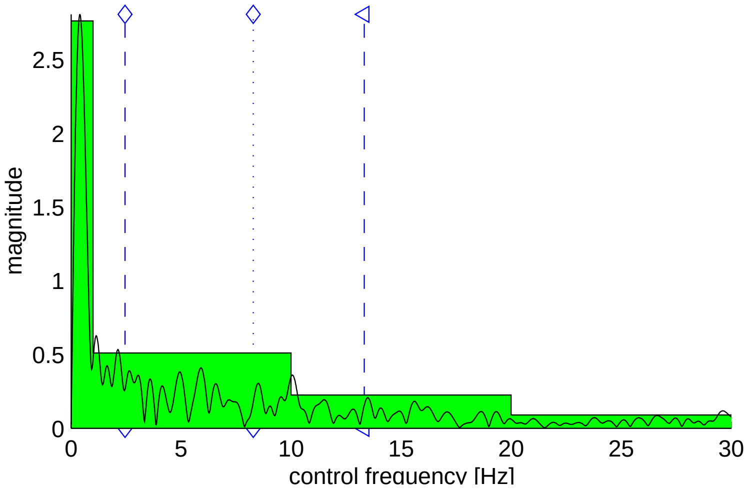
Standard Deviation
: The standard deviation gives an indication of the concentration of the descriptor around the mean.

Minimum, Maximum, Absolute Range
: These range characteristics help us knowing the extent of the descriptor variation.

Start and End Values
: The value at the start and the end of the unit is very useful to calculate the concatenation quality, for example for dinote synthesis (see section 17.1). In order to smooth out glitches or remnants of bad segmentation, we calculate the median of the descriptor values x_i in a short window of the first and last m values, respectively. The window size m is given by a time window of 100 ms.



(a) value characteristics



(b) descriptor spectrum characteristics

Figure 11.1: Characteristic values display of the *loudness* descriptor of a unit

11.2 Temporal Characteristics

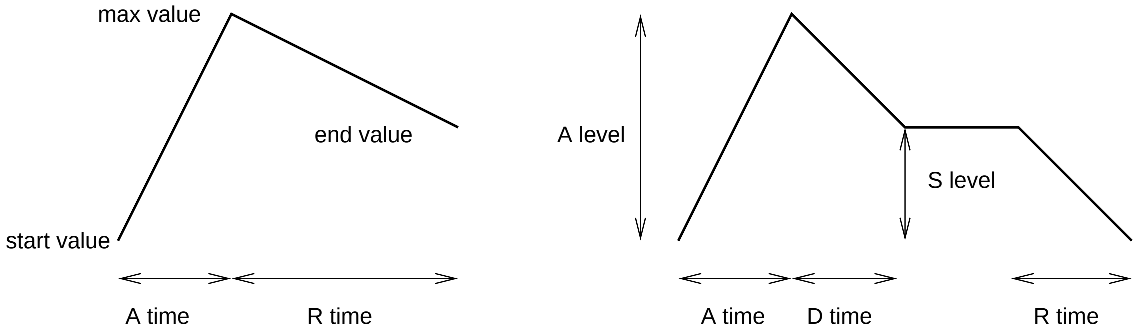
The rough evolution of a descriptor over time is generally called its *envelope*. There are several ways to model the envelope, used concurrently in CATERPILLAR.

AR and Inverse AR Envelope

To fit an AR-envelope (figure 11.2(a)) to a unit, we determine the attack and release time, i.e. the time from the start value of the descriptor to the maximum, and then to the end value. The inverse AR Envelope is goes from the start value to the minimum and then to the end.

ADSR Envelope

One common way to express envelope, used in synthesisers, is called ADSR, from the parameters attack time, decay time, sustain level, and release time (figure 11.2(b)). It can simulate a wide range of acoustic instruments' envelopes. It is modeled by fitting a four-segment curve to the linear energy evolution of the unit (Jensen 1999; Helen and Virtanen 2003)



(a) AR envelope

(b) ADSR envelope

Figure 11.2: AR and ADSR envelopes for temporal modeling of descriptor evolution

Center of Gravity/Antigravity

The temporal center of gravity and antigravity define the location of the most important "elevation" or "depression" in the descriptor curve. They are expressed in normalised distance from the middle point of the unit t_m, i.e. -1 is the start, 0 the middle, and 1 the end of the unit.

The definition of the temporal centroid is the same as the centroid (section 10.6.1), but on the time-domain and not the frequency domain:

$$\mathit{TemporalCentroid} = \frac{\sum_{i=1}^{N} x_i \frac{(t_i - t_m)}{t_m}}{\sum_{i=1}^{N} x_i} = \frac{\sum_{i=1}^{N} x_i \frac{t_i}{t_m}}{\sum_{i=1}^{N} x_i} - 1 \tag{11.1}$$

The temporal center of antigravity or temporal anticentroid is the opposite of the temporal centroid. To calculate it, we mirror the curve about its mean value μ and calculate the centroid as in equation (11.1):

$$\mathit{TemporalAnticentroid} = \frac{\sum_{i=1}^{N} (2\mu - x_i) \frac{t_i}{t_m}}{\sum_{i=1}^{N} (2\mu - x_i)} - 1 \tag{11.2}$$

Polynomial Modeling: Slope, Curve, Residual

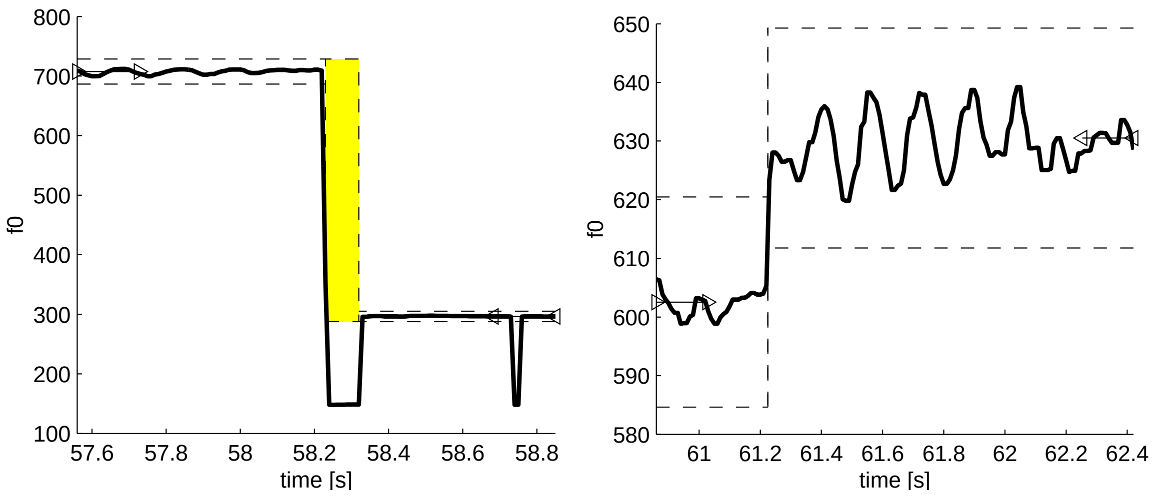
We analyse the average slope, giving the rough direction of the descriptor movement, and the curvature. This serves for more sophisticated concatenation quality estimation, when one wants to assure approximate first and second derivative continuity. These values are calculated by 2$^{\text{nd}}$ order polynomial approximation with Legendre polynomials, which have the desirable property that the lower-order polynomials are valid, albeit more coarse, approximations of the curve. See section 11.4 for more details.

The residual of the polynomial approximation gives us an indication how erratic the descriptor behaves.

Transition Width

This characteristic value has been introduced specifically to model the usability of a dinote unit for instrument synthesis. A dinote is a unit stretching from the middle of a note to the middle of the next note, allowing to perform easy concatenation in the stable sustain parts of the notes. Now, because of imperfections of the segmentation, there can be dinote units that are unusable because they contain more than two notes, or trills or chords. To filter these out, we suppose the unit is a perfect dinote, and compute the width of the area where the fundamental frequency changes from the start value to the end value as defined above in section 11.1. To allow for vibrato, we define a corridor of a width of one halftone around the start and end value. Scanning from the start and end of the unit, the first frame where the raw descriptor leaves this corridor determines the start and end of the transition area, respectively. See figure 11.3 for examples of transitions and the corridors. To allow for the typical error of pitch detection that there are spikes in the result, the descriptor is median-filtered. As can be seen in figure 11.3(a), the spike at the end of the unit is ignored correctly. For the violin corpus, a dinote is suitable for selection when the transition width is smaller than 10 ms. Over 90% of the units in the corpus fulfill this requirement.

As this characteristic value makes only sense for fundamental frequency, it is stored as a descriptor *TransitionWidth*, to save space. Its mean value is the transition width, the start and end value are the begin and end of the transition, and the min and max value are the boundaries of the corridors.



(a) Wide transition

(b) Zero-width transition

Figure 11.3: Transition area and descriptor corridors (dashed) around start/end value (arrows)

11.3 Descriptor Spectrum Characteristics

Similar to the polynomial temporal modeling, but in a different manner, the characteristics of the spectrum of the descriptor curve reveal if the descriptor has rapid or slow movement, or if it oscillates.

To this aim, we calculate the magnitude of the Fourier transform of the descriptor curve, to obtain the frequency components of the *control change*. See figure 11.4 for a schematic demonstration of this.

Spectral Mean, Spectral Standard Deviation, Spectral Skewness, Spectral Kurtosis
The first 4 order centered moments of the spectrum of the descriptor curve tell us if the descriptor is static or has an oscillation, and the regularity of the oscillation.

Spectral Bands
The normalised Fourier spectrum of the descriptor in 5 bands describes the oscillation precisely, e.g. vibrato for pitch will show in the band of 1–10 Hz. The border frequencies are 0, 1, 10, 20, 40, and 100 Hz.

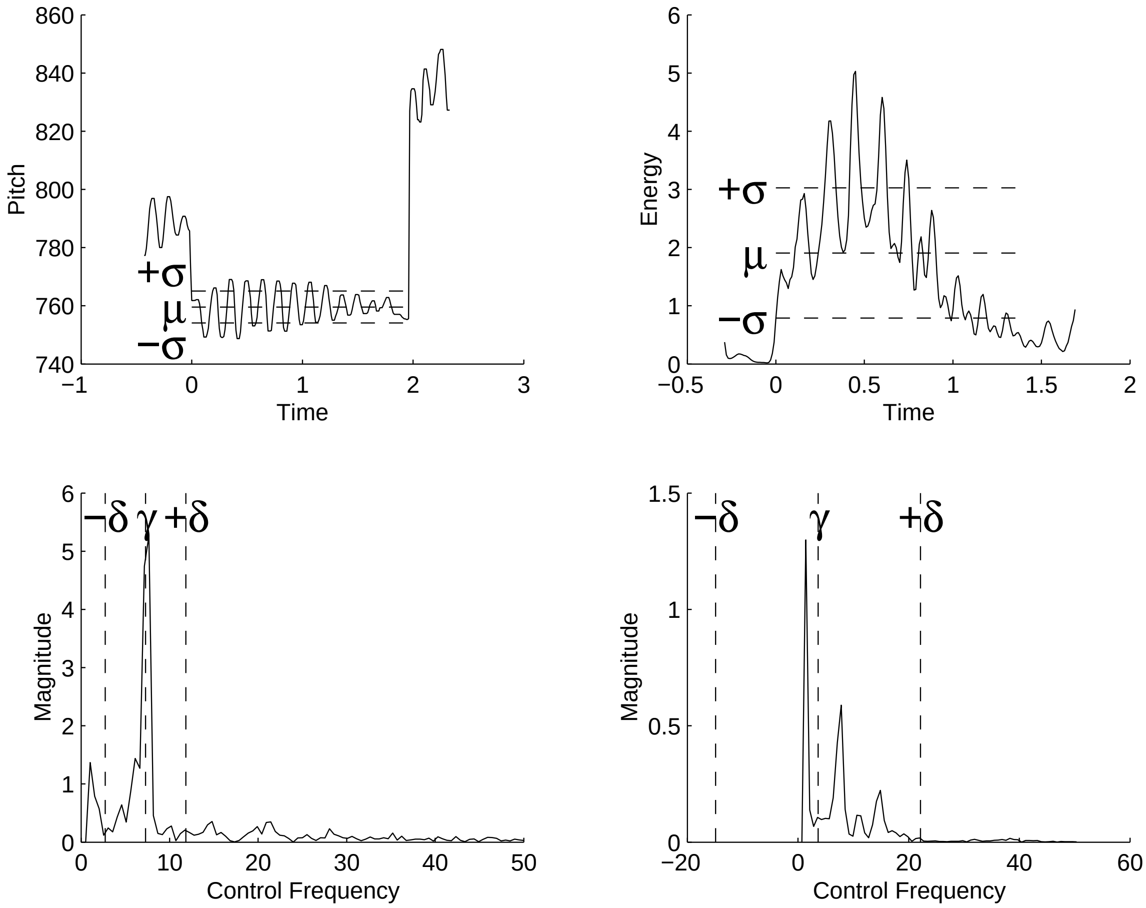


Figure 11.4: Schema of spectrum characteristics of a dynamic descriptor: The raw features (pitch and energy) result in mean μ and standard deviation σ over the duration of the unit, indicated by the length of the dotted lines (top), and magnitude spectrum of the feature, spectral centroid γ, and second order moment δ (bottom).

11.4 Legendre Polynomials

Legendre polynomials are ideal for polynomial modeling of data, because, as will be seen later in detail, they form an orthogonal set. This means that a curve can be approximated to any degree of detail by adding higher order Legendre polynomials. We only need to store n Legendre coefficients to represent all polynomial approximations up to order n.

Canonical polynomials don't have this property, which means that, starting from an n^{th} order approximation, we can not just add the $n + 1^{\text{st}}$ coefficient, but have to recalculate and store all n of them.

In our case, the 2^{nd} order parabolical Legendre approximation contains the linear 1^{st} order approximation by just dropping the highest order coefficient. This idea has also been used in (Rodet and Tisserand 2001) for a scalable representation of descriptor curves over arbitrary length sound segments.

We will now present the mathematical details of the construction and properties of Legendre polynomials:

The Legendre polynomials $P_n(x)$ (Bouvier and George 1983) can be expressed by *Rodrigues' formula* (Weisstein 2003) as

$$P_n(x) = \frac{1}{2^n n!} \frac{d^n}{dx^n} \left(x^2 - 1\right)^n \qquad \text{with } n = 0, 1, 2, \ldots \tag{11.3}$$

or recursively:

$$\begin{aligned}
P_0(x) &= 1 && (11.4)\\
P_1(x) &= x && (11.5)\\
P_{n+1}(x) &= \frac{2n+1}{n+1}xP_n(x) - \frac{n}{n+1}P_{n-1}(x) && (11.6)
\end{aligned}$$

which yields

$$\begin{aligned}
P_2(x) &= \frac{1}{2}\left(3x^2 - 1\right) && (11.7)\\
P_3(x) &= \frac{1}{2}\left(5x^3 - 3x\right) && (11.8)\\
& && (11.9)
\end{aligned}$$

On the interval of $-1 \leq x \leq 1$, they form a complete orthogonal set:

$$\int_{-1}^{1} P_m(x)P_n(x)dx = \begin{cases} 0 & \text{if } m \neq n \\ \frac{2}{2n+1} & \text{if } m = n \end{cases} \tag{11.10}$$

so that a continuous function $f(x)$ can be approximated by a sum of Legendre polynomials $f'(x)$ up to order m (if $m = \infty$, $f'(x) = f(x)$)[1]:

$$f'(x) = \sum_{n=0}^{m} C_n P_n(x) \tag{11.11}$$

with the *legendre coefficients* C_n calculated by

$$C_n = \frac{2n+1}{2}\int_{-1}^{1} f(x)P_n(x)dx \tag{11.12}$$

This property is used in the CATERPILLAR system to express the temporal evolution of a descriptor over a synthesis unit in an efficient way. We chose to approximate the descriptor's curve with a polynomial of order 2 (parabolic). However, we want that all the intermediate approximations (order 0 and 1) to be valid approximations of the curve as well (linear and mean value).

This is not the case for a standard polynomial curve fitting, such as with MATLAB's `polyfit` function: It takes as entry a vector of values x_i and the corresponding $f(x_i)$ and yields the polynomial coefficients $a_1 \ldots a_3$ such that

$$f_2(x) = a_1x^2 + a_2x + a_3 \tag{11.13}$$

is a least mean square error parabolic approximation of $f(x)$. However, $f_1(x) = a_2x + a_3$ is not a good linear approximation of $f(x)$.

11.4.1 Polynomial to Legendre Conversion

The Legendre polynomials do have this property, however, we prefer not to use equation (11.12) to calculate the Legendre coefficients, but split the approximation or curve-fitting method from the polynomial representation. To this end, we developed the conversion from polynomial coefficients $a_1 \ldots a_3$ as for equation (11.13) to Legendre coefficients $p_1 \ldots p_3$ such that

$$l_2(x) = p_3P_2(x') + p_2P_1(x') + p_1P_0(x') \tag{11.14}$$

with x' the normalised x values given by

$$x' = 2\frac{x-\mu}{\rho} \tag{11.15}$$

[1] This is known as a *Fourier–Legendre Series* or *Generalised Fourier Series* orthogonal expansion.

with μ the mean and $\rho = \max(x) - \min(x)$ the range of x.

The mapping is then:

$$p_3 = \frac{\rho^2}{6} a_1 \tag{11.16}$$
$$p_2 = \mu\rho a_1 + \frac{\rho}{2} a_2 \tag{11.17}$$
$$p_1 = \left(\mu^2 + \frac{\rho^2}{12}\right) a_1 + \mu a_2 + a_3 \tag{11.18}$$

In matrix form, the conversion is done by $p = La$ with

$$L = \begin{pmatrix} \mu^2 + \frac{\rho^2}{12} & \mu & 1 \\ \mu\rho & \frac{\rho}{2} & 0 \\ \frac{\rho^2}{6} & 0 & 0 \end{pmatrix} \tag{11.19}$$

11.4.2 Scaled Coefficients Conversion

The MATLAB function `polyfit` described above works best on scaled x-ranges $\hat{x}$ because of numerical stability. The resulting coefficients $\hat{a}$ can be reconverted to polynomial coefficients a usable in equation (11.13). With

$$\hat{x} = \frac{x - \mu}{\sigma} \tag{11.20}$$
$$\hat{a} = \texttt{polyfit}(\hat{x}, y, 2) \tag{11.21}$$

the conversion is performed by $a = B\hat{a}$ with

$$B = \begin{pmatrix} \frac{1}{\sigma^2} & 0 & 0 \\ \frac{2\mu}{\sigma^2} & \frac{1}{\sigma} & 0 \\ \frac{\mu^2}{\sigma^2} & \frac{\mu}{\sigma} & 1 \end{pmatrix} \tag{11.22}$$

11.5 Default Characteristic Values

We have now seen the methods to characteristise a time-varying dynamic descriptor into characteristic values. This could apply also to the constant static descriptors that provide just the mean value for a unit, and to class membership expressed as boolean descriptor, but would constitute a considerable overhead for calculation and storage in the database.

We chose thus not to store the characteristic values for static descriptors in the database, but have them be generated by the database when they are accessed by providing default values. Because of the constant nature of the descriptor values, the characteristic values are easy to define: they are either constant or the unit's mean value. They are given in table 11.1.

This works for note units and the subsegmented units (attack, sustain, release, seminotes), but not for dinotes (see section 17.1.1): Here, we generate a step-curve and have it analsed for the characteristic values. Although it is possible to give analytic formulas for default values known the lengths of both participating seminotes, it is not feasibly integrateable in the database, at least for on-the-fly calculation.

Characteristic Value	**Database Attribute**	**Default Value**
Arithmetic mean	mean	mean
Geometric mean	geomean	mean
Standard deviation	std	0
Start value	startval	mean
End value	endval	mean
Mininum	minval	mean
Maximum	maxval	mean
Absolute range	absrange	0
Slope	slope	0
Curve	curve	0
Residual	residual	0
Temporal mean	t_mean	0 (middle of segment)
Temporal antimean	t_antimean	0 (middle of segment)
Temporal standard deviation	t_std	0
Temporal skewness	t_skewness	0
Temporal kurtosis	t_kurtosis	0
AR Attack time	t_attack	0
AR Release time	t_release	0
Inverse AR Attack time	t_invattack	0
Inverse AR Release time	t_invrelease	0
ADSR attack time	t_A_time	0
ADSR attack	t_A_level	mean
ADSR decay time	t_D_time	0
ADSR sustain level	t_S_level	mean
ADSR sustain time	t_S_time	1 (normalised unit duration)
ADSR release time	t_R_time	0
Spectral mean	s_mean	0 (flat: all energy at 0 Hz)
Spectral standard deviation	s_std	0
Spectral skewness	s_skewness	0
Spectral kurtosis	s_kurtosis	0
Spectral band 0	s_band0	1 (flat: all energy at 0 Hz)
Spectral band 1	s_band1	0
Spectral band 2	s_band2	0
Spectral band 3	s_band3	0
Spectral band 4	s_band4	0

Table 11.1: Characteristic values and their defaults for static descriptors

Chapter 12

Database

The database is the core of any data-driven synthesis system. Its design and capabilities determine the possibilities of the whole system and the paths of future extension. In general, not only the data needed for synthesis, but everything that is persistent between synthesis sessions should go into the database. The CATERPILLAR database holds references to the original sound files and to the data files from analysis. It stores the units and the unit descriptors, the analysis methods and the analysed descriptors, and the relationships between them.

This chapter reports, after a general introduction to relational databases in section 12.1, some issues in designing the CATERPILLAR database in section 12.2. The next chapter 13 describes the interface used to access the database, and chapter 14 details the development and structure of the database. Finally, chapter 15 gives some examples of its contents.

12.1 Introduction to Relational Databases

Relational databases were developed in the 1970s at the IBM research laboratory San Jose (Codd 1970). They marked a break-through over the existing hierarchical and network databases to such an extent, that the latter were completely obsoleted. In fact, nowadays, "database" is synonymous to "relational database". This section will explain why they are such a success. For a general introduction and more arguments pertaining to their advantages over the other methods, see Date (1981).

Relational databases build on the groundwork of the mathematical theory of *relational algebra* and *relational calculus*, themselves applications of set theory, that allowed for the first time to define a simple, consistent framework to describe data and data manipulations, and to operate sound reasoning on it.

In the terminology of relational algebra, the data is contained in a *relation*, which is a set of *tuples* containing as atomic elements *attributes*. It defines the operations *projection* (to select only certain attributes), *selection* (to choose certain tuples), and *join* (to link two relations according to the value of one attribute). The result of a join is again a relation (which satisfies the important algebraical property of *closedness*), with each of its tuples the concatenation of the tuples of the two joined relations.

This theory is embodied in SQL (Date and Darwen 1997), the *structured query language*. SQL is a declarative language that allows to define, populate, and query data in a relational database. Its terminology is somewhat more concrete: A *database* is a collection of tables, views, rules, procedures, and administrative data. A *table* (the relation from relational algebra above) is a set of *rows* or *records* (the tuples), the attributes being *columns* of the table. One attribute or set of attributes form the *primary key* of a table, which must be unique, i.e. no two tuples can have the same value of these attributes. The primary key of each table is *indexed*index, i.e. an efficient lookup-structure is built and kept up to date automatically by the DBMS, to find the tuple with a certain key without searching through the whole table. Indices can be built for any attribute and combination of attributes, if a table is often accessed via these attributes.

Tables can be linked by *foreign keys*, i.e. one table contains as attribute(s) the primary key of another table, such that each tuple *references* a tuple in the other table. Note that this resembles pointers in procedural languages with the difference, that the reference is made to a data value in a referenced tuple, i.e. to its contents and not its location.

A *query* can join multiple tables according to the values in some of their attributes and a comparison operator, and select certain rows and columns from the result. Sorting, grouping and aggregation of rows (count, mean, maximum, etc.) can be specified in the query. A query result is itself a table (an important property from relational algebra). Examples for SQL queries using the CATERPILLAR database can be found in section 14.3. *Views* are shortcuts for often used queries that are assigned a name. They can be used just like tables to be the source of new queries or views. Queries can also change the database by adding or altering table definitions, or inserting or removing data.

The execution of queries is grouped into work units called *transactions* (Gray and Reuter 1993). By default, each query runs in a transaction of its own. Transactions are an achievement of research in *information systems*, e.g. airline reservation systems, where the queries from a multitude of concurrent users need to be serialised and isolated from each other. To always leave the database in a consistent state, transactions have to conform to the ACID principle, the letters standing for:

Atomicity
A transaction is either completely executed or not at all.

Consistency
During a transaction, there can temporarily be an inconsistent state in the database, but at the end, a consistent state, preserving all integrity conditions, is restored.

Isolation
A transaction does not see the intermediate state of other transactions, and none of its changes are visible to the outside before it ends. In short, it runs “as if there were no other”.

Durability
At the end of a transaction (after successful commitment), its effects are persistent, i.e. permanently applied to the database.

12.1.1 Advantages of Relational Databases

As the quality of the synthesis grows with the size of the sound database, an efficient and reliable architecture is required. This is provided in CATERPILLAR by using the open-source relational database management system (DBMS) POSTGRESQL (The PostgreSQL Global Development Group 2002a). Although a relational DBMS results in an overhead for data access, and thus a slight performance penalty, the advantages in data handling prevail:

Data Independence
Using a relational database, only the logical relations in the *database schema* are specified, not the physical organisation of the data. It is accessed using the declarative query language *SQL*, specifying *what* to do, not *how* to do it. This leads to unprecedented flexibility, scalability and openness to change.

Consistency
The consistency and integrity of the data is assured by the concept of atomic *transactions*, which are either completely executed, or rolled back to the previous state of the database, if an error occurred. This means no intermediate inconsistent state of the database is ever visible. This is also an enormous advantage while developing, because programming errors can’t corrupt the database.

Other safeguards are the consistency checks according to the relations between database tables given by the schema, and the automatic reestablishment of referential integrity. For example, in the CATERPILLAR database, the units belonging to a sound file are automatically deleted when the sound file is deleted, triggered by a so-called *foreign key constraint*, given by specifying that the attribute bfid in table Unit (A.2.3) references, i.e. contains the primary key of, table BaseFile (A.2.1), and that deletions of a referenced tuple should be cascaded, i.e. that the referencing tuple in Unit should be deleted as well.

These constraints are installed in the table creation statement given here in SQL:

```
CREATE TABLE Unit (
        uid     integer PRIMARY KEY,
        bfid    integer REFERENCES BaseFile ON DELETE CASCADE,
        ...
);
```

Additional consistency constraints not covered by the automatically performed checks can be specified using *trigger procedures*: These are functions[1], called upon insertion, update or deletion of a tuple in a database table, that can enforce these application dependent constraints. For example, in CATERPILLAR, we stipulate that any sound file added to the base have a representant unit covering the whole file. This constraint is realised by a trigger procedure that adds the representant unit automatically, each time a sound file is added (see tables Unit (A.2.3) and BaseFile (A.2.1) explained in the database schema). The advantage is here that the responsibility for a consistent state is moved from the applications using the database to the database management system, so that the applications become less complex and error-prone.

Client–Server Architecture

Concurrent multi-user access over the network, locking, authentication, and fine-grained control over user's access permission are standard features of DBMS.

At first, this seems to be not so much an issue for a research prototype or a personal composition system such as CATERPILLAR, but I made the experience that while adding large amounts of sounds, which can take a while, it is a great relief that the system is not blocked by this task. Thanks to the ACID principle, one can indeed access the database while data is being added, and will never see an inconsistent state (just an incomplete one for the new data as sound files, units, and features are being added).

12.1.2 Database Schemas

A database is defined by its *database schema*. It specifies the tables and views in the database, the relationships between them, and their attributes and their types. Possible integrity constraints are also given in the schema.

The database schema is written in SQL, and can be represented graphically like a UML class diagram (Rational Software 1997; Martin 1997; Booch, Rumbaugh, and Jacobson 1999), as for example in figure 14.5. However, to abstract from the concrete implementation issues, and to reason on the level of a model of reality, the schema is usually first elaborated in Entity/Relationship (E/R) notation. See figure 14.1 for an example.

The rectangles are the entities (which can be adorned by their attributes but are left clean here for clarity). The lozenges (diamonds) are the relationships between two or more entities, notating explicitly the cardinality of the relationship: 1/1 for a one-to-one relationship, 1/+ for a one-to-many relationship, and +/+ for a many-to-many relationship. Instead of "1" or "+", we can use "?" or "*" to specify a cardinality of 0 or 1, or 0 or more, respectively. This means there can also be no members in the relationship (on one or both sides).

When an Entity/Relationship model is implemented in an SQL schema, an entity maps naturally to a table. According to its cardinality, a relationship is implemented either using a foreign key reference, for one-to-one or one-to-many relationships, or as a table containing the two foreign keys

[1]The database functions, also called *stored procedures*, can be programmed in SQL or any programming language that can be called from the database, such as the POSTGRESQL-specific procedural extension to SQL, *PL/pgSQL*, *Python*, *Perl*, or C

of the related entities for many-to-many relationships. If zero cardinality is allowed, this means that the foreign key can be NULL.

12.2 Some Modeling Issues with Relational Databases

As in all software systems, a relational database schema models a section of reality. Here, it is constituted by the manifest real world objects and their relationships: sound files, categories, units, descriptors, analysis methods, etc. However powerful, the relational paradigm does not allow for some of the informational structures of reality to be modeled directly.[2] What's more, database development is a two-step modeling: First, reality is modeled abstractly in an Entity/Relationship model. Second, the implementation of the E/R model in a concrete database schema can itself be seen as a modeling, because, again, the E/R diagram can not be transferred one-to-one to the database schema.

12.2.1 Modeling Inheritance

One such informational structure to be modeled and stored by the database is the generalisation/specialisation of classes, also called *inheritance*. It appears for instance when modeling references to files, e.g. sound files or data files. Evidently, these two have many things in common: they both need a path and filename to be stored, and the date of last modification. Then, there are other attributes that are different for the two: The sound file needs information about the sampling rate and the sample format, the data file information about the data encoding (ASCII or binary) and the data format. Using inheritance, we would define a base class with the attributes common to sound and data file, and two derived classes containing the information specific to the two.

We can indeed model this inheritance with E/R modeling, using the generalisation/specialisation relation, a large triangle pointing to the base class, as can be seen in figure 14.2. However, it is not possible to express it in standard SQL. Some relational DBMS, among which PostgreSQL, used for the implementation of the Caterpillar database, have extended SQL to allow inheritance between tables, but these extensions are non-standard, and sometimes awkward. This is why in the actual database schema, inheritance is implemented by using the following trick: We define a *fat base class*, i.e. a table that contains all the attributes of all the derived classes, and a selector attribute that tells us which derived class an entry belongs to. The derived classes are then implemented by views that project only the attributes needed for the derived class.[3][4] Insertion of data has still to be done on the base class, but can be handled by *stored procedures*, i.e. subroutines of SQL-statements defined with the database schema and stored in the database.

Examples of the use of inheritance in Caterpillar are the tables Descriptor (A.1.1) and BaseFile (A.2.1) and their derived views.

12.2.2 Representation of Class Hierarchies

The last section talked about modeling an inheritance relationship between database tables, i.e. entities. In this section we'll study how to model any sort of hierarchy between data elements, i.e. tuples, in our tables.

Using tree structures in a relational database is not straightforward. It is no problem to model a tree structure with references to the same table, but querying these is a problem, since the declarative

[2] To remove this shortcoming, object-oriented databases have been developed, but they lack the strength of simplicity of relational databases (being based on relational algebra) and reintroduce some problems of 1960's hierarchical or network databases whom to solve relational databases were developed in the first place!

[3] The attributes of the other derived classes should be NULL for this entry. This reveals possible disadvantage of this method: We cannot define a NOT NULL attribute for derived classes.

[4] Another possibility is to create a table with the base classes attributes only, and one table per derived class, and to link them via the primary key of the base class. (This resembles very much the implementation of inheritance by pointers in C++.) This solution has not been adopted for Caterpillar to keep the schema simple, as there are already many tables in it.

language SQL does not provide the needed loop construct to search upward or downward through the tree. To come around this limitation, in CATERPILLAR, we keep the transitive closure of the tree, i.e. we store the link of a category with its direct base class, and the links to all further base classes. This also results in faster access, because all base categories can be queried immediately.

12.2.3 Representation of Categorical Descriptors

Categorical descriptors have non-continuous, non-numerical values, which can be either binary, or multi-valued as an enumeration of several categories. Examples of categorical descriptors are the membership of a database unit in a class of a hierarchy, like *source* (with simultaneous membership in *violin*, *strings*, *instrument*, for example), or categories such as *phoneme* or *unit type* (exclusive values *note*, *attack*, or *region*, for example).

There are several possible approaches to this:

- We can store the name of the category that a unit belongs to as a text value of a descriptor *category*. We see easily that this is not practical, because a unit can be in many categories, and, for class categories, to find out the membership in the base classes we would have to do a search in the class hierarchy.
- We can use as descriptor f_c for category c the *characteristic function* $\chi_c(u)$, which is 1 for a unit u only if the unit belongs to the category, and 0 otherwise. This has the advantage that it can be extended to a *fuzzy* function, giving the "degree" of membership to a class by using a value between zero and one. However, definition and storage of these descriptors is not feasible, since all existing categories (and all categories added later) would need to be represented in all units.
- The solution adopted in CATERPILLAR is not to store class membership as a descriptor, but to generate it on the fly as a boolean value, also for all base classes present at the time of the query, as described in section 14.2.4.

  Category descriptors without class structure (unit type, source sound file) are stored as an enumeration value: an integer out of a fixed list, mapped to a text value. (This mapping is stored in the table Symbol (A.1.2).) Some categories are conveniently expressed using class membership *and* enumerated values: The category *phoneme* is best represented as an enumerated descriptor of the same name, and as membership in one class per possible phoneme, which are the leaves of a phoneme hierarchy (see figures G.2ff).

Chapter 13

Database Interface

The CATERPILLAR database is clearly separated from the rest of the system by a *database interface*, as depicted in figure 13.1. Therefore, the DBMS used can be replaced by a different system, or other existing sound databases can be accessed, e.g. using the MPEG-7 indexing standard, or the databases built as results from the CUIDADO or ECRINS projects described in chapter 2.2.

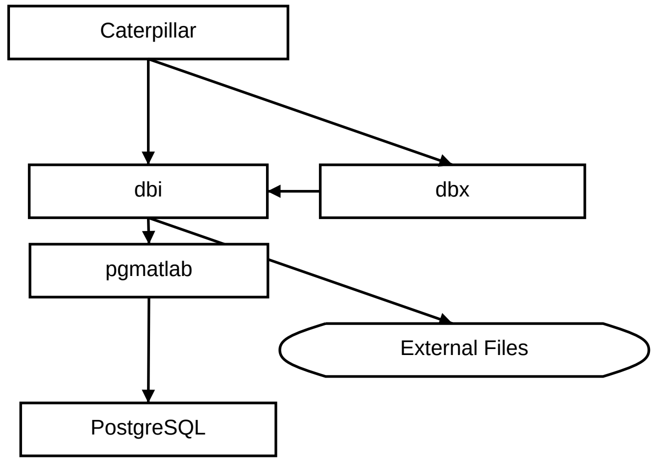


Figure 13.1: Structure of the database interface.

At the bottom, the relational DBMS POSTGRESQL (The PostgreSQL Global Development Group 2002a), released as open source free software, is reliably storing hundreds of sound and data files, tens of thousands of units, their interrelationships and descriptor data. It is accessed by the low-level functions of the *pgmatlab* interface described in section 13.1. Section 13.2 presents some conceptual differences occurring in the access.

The database interface is embodied by the MATLAB-functions *dbi* and *dbx*, described in section 13.3, and handles also access to external files (section 13.4)

13.1 Low-level Database Access Functions

For low level access to the relational database from MATLAB, we wrote *mex*-extensions, i.e. new commands for the MATLAB environment written in C, that use the *libpg* client library to access the POSTGRESQL database server. These functions of general usefulness are released as the open source software project *pgmatlab* under the Gnu Lesser General Public License (LGPL). The project is hosted by *GBorg*[1].

The main functions are:

[1] http://gborg.postgresql.org/project/pgmatlab/

psqlconnect
Connect to or disconnect from a database server on the local or a remote host.

psqlconfig
Set/get configuration variables for the MATLAB/POSTGRESQL interface.

psqlquery
Execute an SQL query on the current POSTGRESQL database. The result is a double matrix if all the values in the query result are numerical and non-empty (not NULL), or a cell array otherwise. The error code and the list of the resulting column names is optionally returned.

psqlalter
Execute SQL query to insert/update/delete on database. Returned are the status and optionally the unique object id of the inserted row.

psqlcall
Call database function written in the procedural SQL extension language *PL/pgSQL*.

There are several other small helper functions, mainly to prepare data to be used in queries, like to perform the necessary quoting of special characters in SQL strings.

13.2 Mapping SQL to Matlab

In general, SQL's basic data structure which is the table (with a different data type for each column) maps nicely to a MATLAB cell array (with a different data type per element). The SQL NULL value, which stands for "no data" maps naturally to an empty cell. The special case of only numeric values is mapped to the basic MATLAB data type of numeric matrix. Here, NULL is represented as the not-a-number (NaN) value. The transition is much more transparent than for traditional procedural languages without operations on complex data types like matrices, which are "row-at-a-time", needing to introduce into SQL the concept of cursor, that can be used in a loop to retrieve (fetch) one row at a time. MATLAB and SQL are both "set-at-a-time".

However, there's a subtle paradigm shift between SQL and MATLAB, that sometimes breaks the transparency of the transition, when care is not taken. The problem is what identifies uniquely a tuple. In SQL, we have the primary key as unique identifier, i.e. a part of the contents of a row, independent of its position in the table, and not necessarily in sorted order. In MATLAB, we identify a matrix row by its index, i.e. its position in memory. The problem occurs, for example, when the data of a set of units given by their uids is to be loaded into memory. To avoid having to find the corresponding row by the unit id each time we access a unit, we can assure that the data is in the same order as the requested uids, but it can happen that no data is present for a unit. In SQL, there will naturally be no tuple for this unit. In MATLAB, we need the row to be present at its expected index with not-a-number values. We can avoid this problem by using MATLAB's *sparse matrices* for unit data, where only non-zero elements actually take space, indexed by the uid.

13.3 The *dbi* and *dbx* Functions

The MATLAB function *dbi* (for *database interface*) defines the high-level application dependent calls to the CATERPILLAR database. It also encapsulates some external data or sound file access for convenience (see section 13.4). The details of the *dbi* sub-functions are given in appendix B.

Only this function would have to be rewritten when a different database is to be used for synthesis, such as the *Studio On Line* sound database (Wöhrmann and Ballet 1999).

If, for whatever reasons, the underlying DBMS is to be changed, only the low-level access functions would have to be adapted, or rewritten to access the new DBMS, since they are POSTGRESQL specific. The queries issued by *dbi* are fairly standard SQL, excepting some syntactic differences. A little bit of work would have to be done on the stored procedures written in *PL/pgSQL*, but every reasonable DBMS has a structurally similar procedural language. (To illustrate this, the PostgreSQL programmer's documentation (The PostgreSQL Global Development Group 2002b) contains a chapter with clear steps how to translate from ORACLE's procedural language and back.)

The *dbx* function (for *database explorer*) gathers all functionality for displaying the data in the Caterpillar database, and browsing its contents. It relies in turn on *dbi*. Its sub-functions are given in appendix C.

13.4 External File Formats

The external sound files can be in any format (aiff, wav, sf, au, etc.), and are unified and handled by the Matlab class *SoundFile*

The Sound Description Interchange Format (SDIF) (Wright, Chaudhary, Freed, Khoury, and Wessel 1999) is used for well-defined interchange of data with external programs (analysis, segmentation, synthesis). See section 13.4.1 for an explanation of the aims and principles of SDIF. The details of the description types defined for Caterpillar are given in appendix E. The following sections 13.4.2 and 13.4.3 explain a useful selection syntax to access parts of the data in an SDIF file, and programming interfaces for external languages and systems developed in the course of this work.

Other data formats are the common untyped ASCII break-point functions, with time in the first column and value in the second. Some legacy programs still use this format although it should be replaced by SDIF.

13.4.1 The Sound Description Interchange Format (SDIF)

SDIF was developed in 1996–1997 as a collaboration between Ircam—Centre Pompidou[2], the Center for New Music and Audio Technologies (CNMAT)[3], University of California Berkeley, Department of Music, and the Music Technology Group (MTG)[4] of the Audiovisual Institute, University Pompeu Fabra (IUA–UPF), Barcelona.

Its aim was to replace the multitude of badly defined application-specific ascii or binary formats for sound analysis files with a standard that could be used to interchange data between different programs and institutions.

SDIF is in fact a meta-format that allows description types for sound representations, analysis data, and synthesis control data to be defined. There exists a set of standard description types for time domain signals, Fourier transforms, fundamental frequency, sinusoidal partials or spectral peaks, spectral envelopes (Schwarz 1998), resonant filters, and FOFs (see section 4.1.1 and Rodet 1984).

New description types can be defined, and existing types can be extended to carry additional data. These definitions can be made either in the standard type definition file, or in the SDIF data file itself. Thus, a file with new or extended types will still be well-defined, and the semantics of the data known. Programs that don't understand the new or extended description types can always skip or ignore the additional data.

SDIF is a tagged file format initially inspired, via AIFF, from the Amiga IFF. The tags are 4 characters wide and are called *signatures*. The first character specifies the class and version of the type. See appendix E for details. It is organised around time-ordered *frames* which are grouped into *streams* for example for stereo files. A frame contains any number of *matrices* with named columns and any number of rows.

Other work on SDIF concerns applications, connections with other software systems, and extensions and conventions of its use (Wright, Chaudhary, Freed, Wessel, Rodet, Virolle, Woehrmann, and Serra 1998; Wright and Scheirer 1999; Wright, Dudas, Khoury, Wang, and Zicarelli 1999; Wright, Chaudhary, Freed, Khoury, Momeni, Schwarz, and Wessel 2000; Schwarz and Wright 2000).

---

[2] `http://www.ircam.fr`
[3] `http://cnmat.CNMAT.Berkeley.EDU/SDIF`
[4] `http://www.iua.upf.es/mtg/eng`

SDIF and the interface libraries described in section 13.4.3 is published as open source software under the Gnu Lesser General Public License (LGPL) on Ircam's SDIF web site[5] (Virolle, Schwarz, and Rodet 2002; Schwarz 2001). Discussion and support is available on the SDIF mailinglist[6], created and administered by the author.

13.4.2 SDIF Selection

SDIF files can contain aggregates of different sound descriptions for one or multiple sound objects, hence the need to address a subset of these. The SDIF selection, described in (Schwarz and Wright 2000), defines a syntax convention to specify a part of the data in an SDIF file (section 13.4.2.1). For example, certain time ranges, certain streams, certain frame or matrix types, or certain rows or columns of the SDIF matrices can be selected. This selection specification can be conveniently added to a file name, so that command-line programs transparently inherit powerful selection capabilities. They only have to use a selection-aware SDIF library, such as IRCAM's (Schwarz 2001), which takes care of parsing and executing the selection. There are more applications for the SDIF selection syntax, described in section 13.4.2.2: It can choose the output sdif type of programs, e.g. when converting from untyped legacy formats, or map description types to display modes or actions.

13.4.2.1 Selection Syntax

The syntax for a filename, possibly including the directory path, with an SDIF selection is:

[*filename*]`::`[`#`*stream*][`:`*frame*][`/`*matrix*][`.`*column*][`_`*row*][`@`*time*]

The start of the selection specification is marked by the last '`::`' occurring, followed by 6 optional selection elements. This way, there is no ambiguity with filenames containing the selection element markers.[7] The order of the selection elements is not significant, and the selection specification can contain white space. All element specifications can be comma-separated lists of values. Numerical specifications can also be ranges *lower*`-`*upper*, or delta ranges *value*`+`*delta*, selecting the range from *value*−*delta* to *value*+*delta*. The element markers have been chosen to minimize clashes with Unix shell special characters. Their mnemonics and meanings are:

- `#`*stream-id* as with "number" or `$`*stream-id*[8] as with "stream" selects all streams with the given numbers or names.
- `:`*frame*`/`*matrix*, as in a file hierarchy, selects the data with matching signatures. If only the frame element is present, all matrices are selected. If only the matrix element is given, all matrices of that type are selected, independent of the frames they occur in. This allows, for instance, to view all comments in `1CMT` matrices in a file.
- `.`*column* as in a C-struct and `_`*row* as in a LaTeX index select a sub-matrix in the selected matrices.

  Beware that column, and sometimes row and matrix selections can produce invalid SDIF output lacking required columns for a given matrix type or required matrices for a given frame type. However, it is very useful to be able to do this for external analysis of the data in an SDIF file. SDIF tools should check if they are allowed to select columns or rows. If they are, the order of the column and row selection is significant, allowing re-ordering.

  Columns can be given either as a number or as a name. The column names are given by the matrix type definition.

[5] `http://www.ircam.fr/sdif`

[6] `http://list.ircam.fr/wws/info/sdif`

[7] To specify a filename containing '`::`' itself, simply append '`::`', which means an empty selection matching all data.

[8] The dollar sign can be used in URLs instead of `#`, which has a special meaning there. This is to allow selections with an SDIF-aware web server.

- `@`*time*, as in English “at time *t*”, selects the frames in the given time range.

For example, to specify the part of the SDIF file piano.sdif which is contained in stream number 1 in `1HRM` frames and matrices, and to select only columns 3 and 2 (amplitude and frequency) of rows 1 through 50 (the first fifty partials appearing in each matrix), between the times 1.999 and 2.001, one can say:

```
piano.sdif :: #1 :1HRM /1HRM .3,2 _1-50 @2+0.001
```

As a shortcut, if the first selection element is the frame, the frame element marker ’`:`’ can be dropped. So, instead of *filename*`:::`*frame*`@`*cetera* it is *filename*`::`*frame*`@`*cetera*.

13.4.2.2 Applications

In Caterpillar, SDIF selection is used when descriptor analysis programs write output files containing more than one descriptor. The database table Analyses (A.1.5) implements the one-to-many relationship that tells us which descriptors an analysis program outputs and how to access this descriptor’s data in the output SDIF file, using only the selection part of an SDIF selection specification.

For the data-driven inversion of physical models D’haes and Rodet (2001, 2002, 2003), use a database built on SDIF files. To access specific data from the training and classification algorithms, he uses the SDIF selection, which is similar to the use of a DBMS and SQL queries in Caterpillar. It can be argued that the SDIF selection is a very simple subset of an SQL SELECT query, given an appropriate database schema modeling the data in an SDIF file.

The SDIF selection defines a syntax and a standard semantics for programs reading SDIF files: Specifying a file name plus a selection is exactly equivalent to removing all but the selection from the SDIF file and then specifying that new file (which can be empty). Implementing that semantics is made very easy in the Ircam SDIF library by high level functions: Opening a file automatically parses the selection specification, the high-level reading functions automatically find only selected frames and matrices, and there are functions which perform the row and column mapping given by the selection.

All Ircam SDIF tools accept an SDIF selection for reading. This works also with standard input, using as filename `::`*select-spec* or `-::`*select-spec*.

For writing, the selection can be used to specify the output description type. For example, in Ircam’s converter from untyped ASCII data to SDIF, there is a choice between `1HRM` or `1TRC` output for partial data, or the output type of the ubiquitous two-column break-point function files:

```
bpftosdif glass.f0 glass.sdif::1FQ0/1FQ0
```

As these programs also read SDIF, we immediately have an SDIF type converter (use at your own risk).

The selection can also be used to specify the way description types are displayed: Data in any SDIF frame type can be plotted, using the selection syntax to select the matrices and two columns used for the x- and y-axis. For example, `mongol.sdif::/1HRM,1TRC.2,3` as filename would plot partial amplitude over frequency.

The SDIF selection can also be used for mapping actions to description types, e.g.: “Edit this frame type with that program”.

13.4.3 SDIF Interfaces with other Languages and Systems

Matlab
: The SDIF interface for Matlab (Schwarz and Wright 2000) is essential for Caterpillar, since the database interface that handles external files is written in Matlab.

It allows convenient reading and writing of entire MATLAB matrices to and from SDIF files. It is released as open source software on the SDIF web site[9].

jMax

The SDIF reader/writer for the *jMax* real-time interactive sound synthesis environment is used to write the results of HMM alignment to files that can be directly used as segmentation definition for unit import.

It is released as open source software, hosted by *SourceForge* under the *jMax*project[10] or at IRCAM's free software site[11].

Scripting Languages

The *simplified wrapper interface generator* (SWIG)[12] (Beazley, Fulton, Köppe, Johnson, Palmer, Files, Yerkes, and Beckford 2003; Beazley, Fletcher, and Dumont 1998) is a tool that generates from a C-library or C++ class hierarchy bridges callable from a great number of scripting languages, such as Perl, Python, TCL, PHP, and many others.

The Perl interface was used for the scripts that merge various ASCII file types into a canonical SDIF representation of phone data in the artistic speech synthesis application described in section 17.5.

[9] `http://www.ircam.fr/sdif`

[10] `https://sourceforge.net/projects/jmax/`

[11] `http://freesoftware.ircam.fr`

[12] `http://www.swig.org`

Chapter 14

The Caterpillar Database Schema

After the introduction into databases and a brief discussion on how to model reality in a database schema in the last chapter, this chapter describes the actual database schema designed and implemented for the Caterpillar database. The Caterpillar database schema is quite complex. In order to make it accessible for the reader, section 14.1 will first give an overview, making shallow incursions into the schema from various starting points. Section 14.2 gives more details of the design, focusing on various functional aspects. These are illustrated by the examples of some typical data and queries in section 14.3, to motivate the schema from its actual use. The detailed definition of the actual tables and attributes implementing the Caterpillar database schema are given in appendix A. The last section 14.4 shows some paths for future extensions of the schema.

14.1 Overview

An overview of the structure of the Caterpillar database is given in the Entity/Relationship diagram in figure 14.1.[1] It shows the main entities and relationships: The entity Unit represents the elements of the hierarchy ParentUnit and graph NextUnit. Each unit Is In one or more Categories, and is part of a SoundFile. A Category can be a specialisation of another Category, expressed by the inheritance hierarchy Is A. The relationship AnalysisRun records that a SoundFile has been analysed to yield a FeatureFile containing the raw features of one or more FeatureTypes. These have been distilled to a vector of UnitData per unit.

The complete database architecture design is specified by the Entity/Relationship diagrams in figures 14.2 and 14.3. The database design schemas in E/R notation are given in a conceptual and a concrete version. The conceptual version in figure 14.2 reflects all conceptual design decisions, such as generalisation (inheritance). However, in the current DBMS inheritance is not always possible, and if it is, it is not standardised. It can, to a certain degree, be simulated using views as explained in section 12.2.1. That's why the concrete version in figure 14.3 gives the E/R schema of the database that has been actually implemented in database tables.

This implementation in SQL of the E/R model is documented in the diagrams 14.4 and 14.5 in UML notation (*Universal Modeling Language*, see (Rational Software 1997)). These diagrams were generated from the SQL source file by extracting the foreign key references from the table creation statements, and, for views, their dependencies on other tables or views. The automatic graph layout program dot[2] (Koutsofios and North 1996) then finds the best disposition for the edges and nodes.

In the UML schema of the Caterpillar tables in figure 14.4, the arrows show that a table is referencing another one, and with which attribute. This means, that an entry in the referencing table contains as the shown attribute the key of a tuple in the referenced table. Figure 14.5 additionally shows the views as dashed boxes, and the tables they are based on.

[1]In this and the following E/R-diagrams, we adopt the convention that entities and relationships that actually exist as tables in the SQL database schema are capitalised, whereas relationships that are implemented as a foreign key reference are in lower case.

[2]`http://www.graphviz.org`

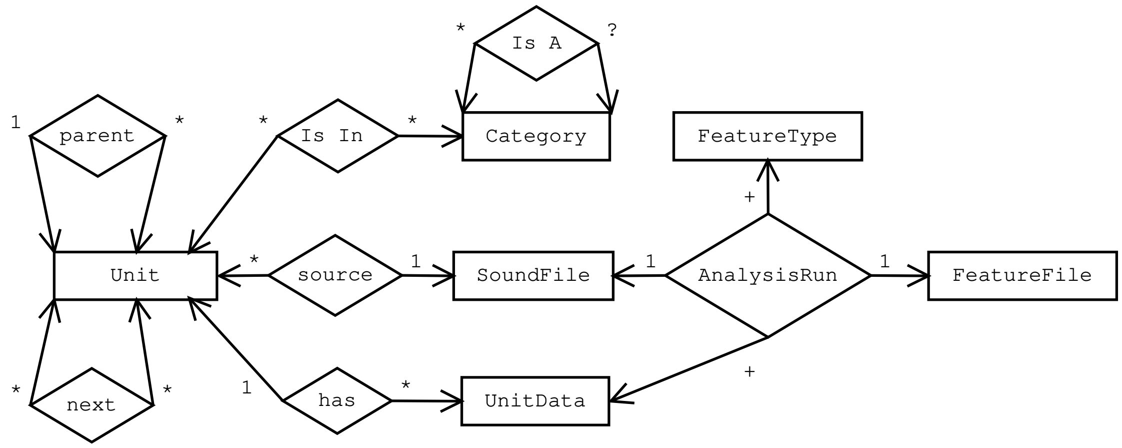


Figure 14.1: Overview of the CATERPILLAR database schema in Entity/Relationship notation, showing the most important entities (rectangles) and relationships (lozenges) with their cardinalities.

We can see three tables that do not reference anything: FeatureAnalysis, BaseFile, and Descriptor. These tables are the manifest entities given from the "outside world". We can also clearly distinguish the three most important tables in the diagram as the ones which are the most referenced: Descriptor, BaseFile, and Unit. AnalysisRun serves as a sort of glue between these tables.

14.2 Details of the Database Design

This section explains the CATERPILLAR database design under various higher-level aspects of its function, which are the sound descriptors and categories (14.2.1), the sound and data files (14.2.2), the units and their relationships (14.2.3), their membership in categories (14.2.4), the special representant units (14.2.5), the descriptor extraction (14.2.6), and finally the unit data (14.2.7). The complete details of the table's implementation can be found in appendix A.

We situate ourselves in between the conceptual modeling in E/R diagrams, and the implementation in SQL tables. Depending on which aspect we describe, we will sometimes refer to entities and relationships, and sometimes to tables. There is no risk of confusion, since each entity has a corresponding table or view.

14.2.1 Sound Descriptors and Categories

The table Descriptor represents one sound descriptor (see chapter 10), which is the "data-type" of the actual unit data. Descriptors are either categories or analysed static or dynamic features.

We can see that the Category, Corpus and FeatureType Entities are specialisations of Descriptor. This is because they can both represent data for selection. FeatureType represents static or dynamic descriptors, and Category categorical or class descriptors. Category is in fact a specialisation of the Corpus entity which groups units into arbitrary collections which one desires to use for synthesis (see chapter 15). The specialisation goes in this sense, because every Category can also represent a Corpus, e.g. when we want to perform a selection from all *violin* units, but the inverse is not true: not any Corpus constitutes a category descriptor. Examples of corpora are: all string units, pieces by Bach, pieces by Bach played by a certain musician, the contents of a sampling CD.

The table IsA defines the inheritance hierarchy of categories or corpora (represented by entries in table Descriptor). The system is open to handle all possible descriptors which can be added or changed dynamically by the user.

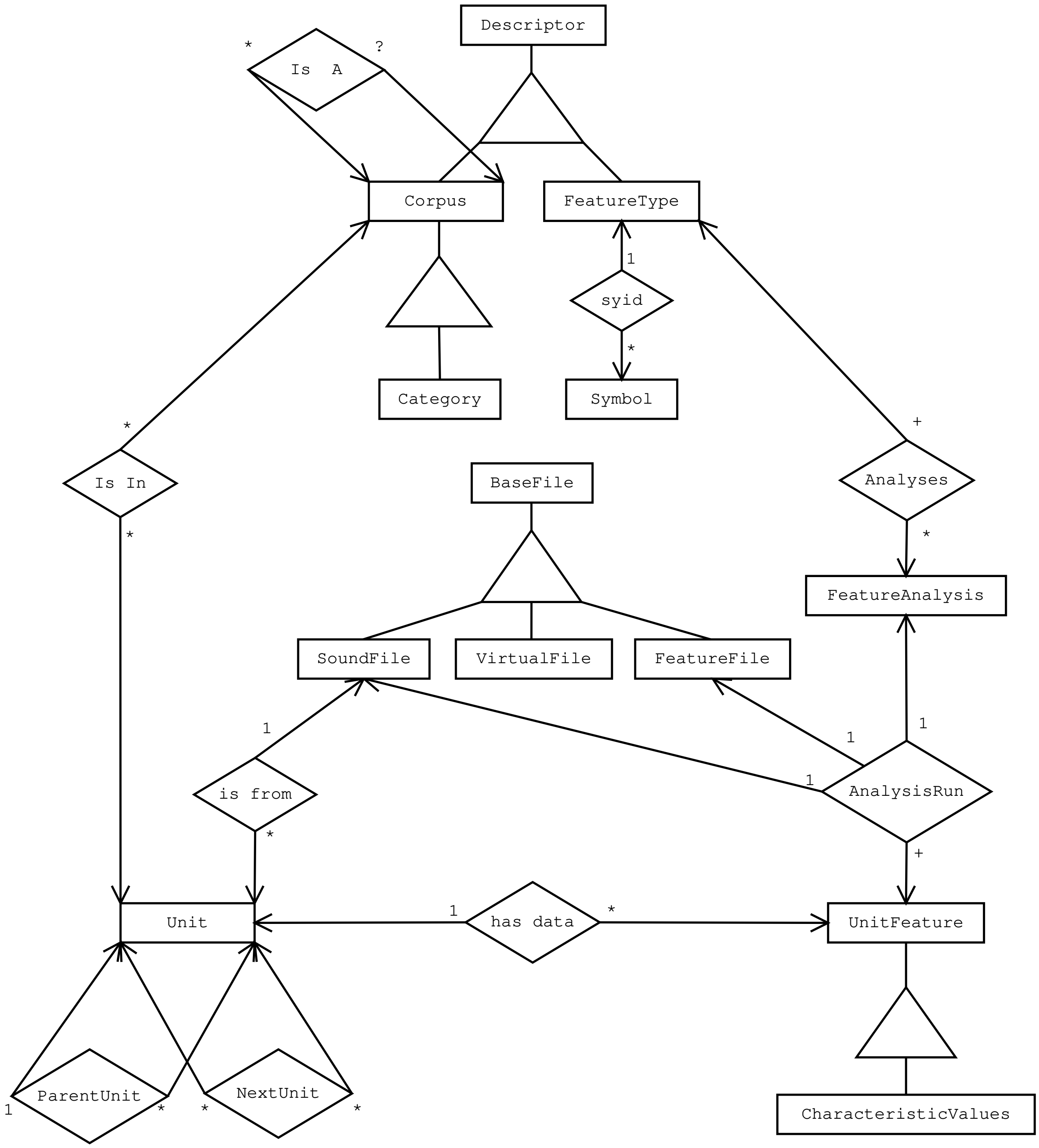


Figure 14.2: Entity/Relationship diagram of the conceptual database schema design

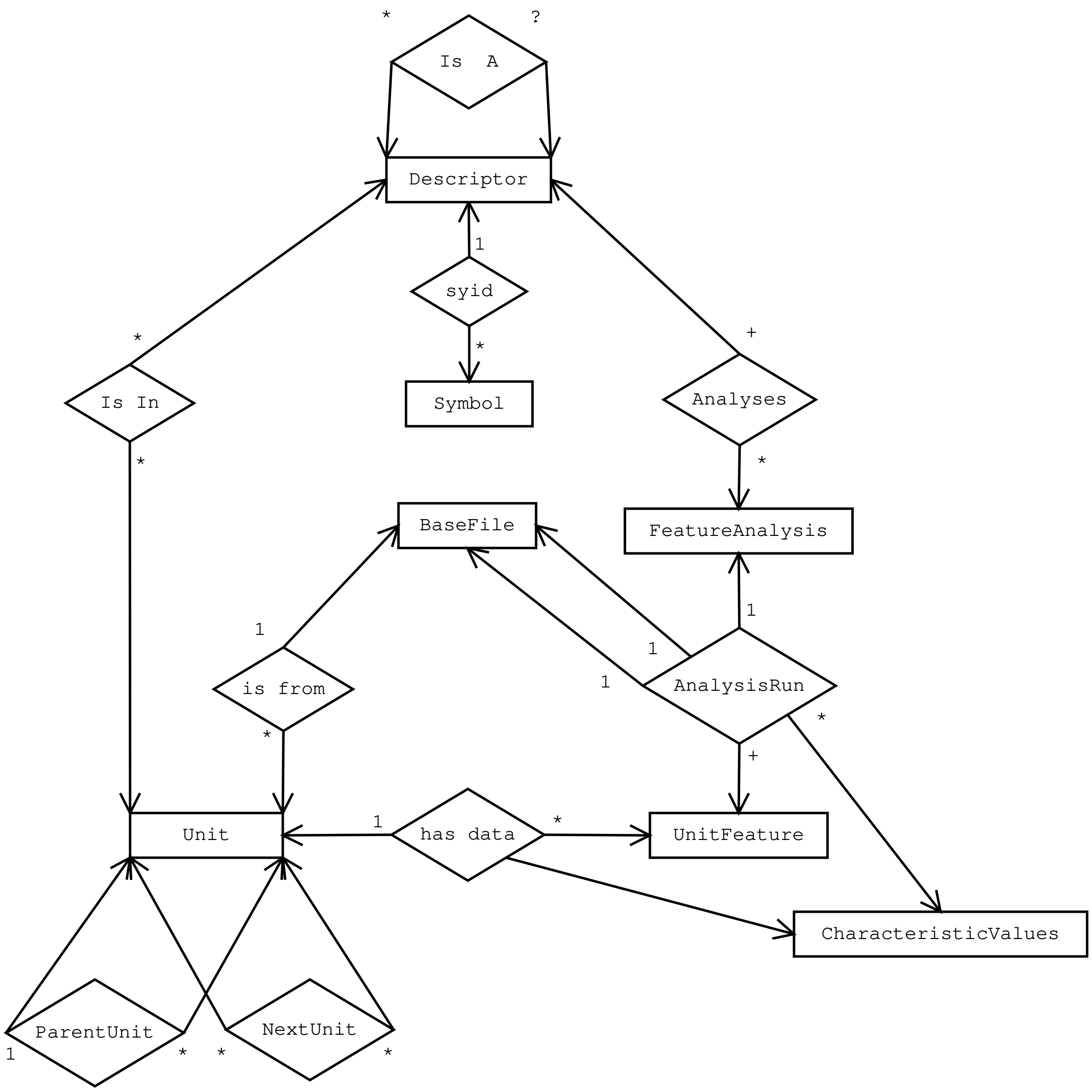


Figure 14.3: Entity/Relationship diagram of the concrete database schema design

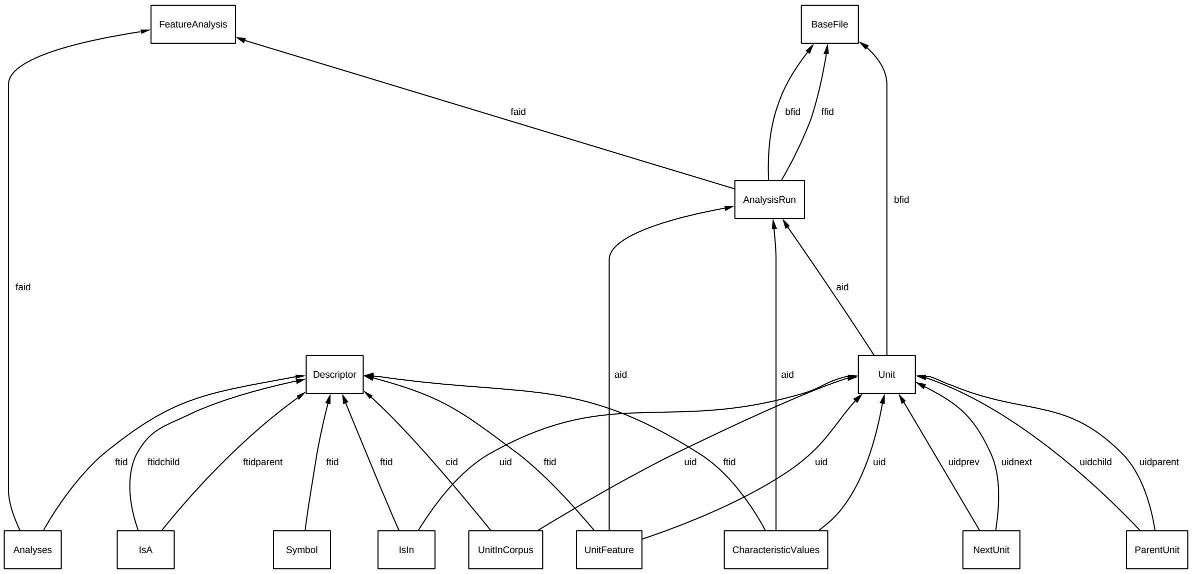


Figure 14.4: Partial SQL implementation schema of the CATERPILLAR database in UML notation, showing only tables and the foreign key references between them.

14.2.2 Sound and Data Files

The table BaseFile contains a reference to a sound file (i.e. its path in the file system) and its attributes, or to a feature file with analysis data. It is convenient to store also the synthesis target in the database. For this, *virtual* files have been introduced that don't reference any external files, but serve as a container for the target with its sequence of units and their data.

14.2.3 Units and their Relationships

The data defining the units to be selected and concatenated are stored in table Unit. This encompasses the sound file the unit is from, the start and end time within, and the type of the unit (*note*, *attack*, *diphone*, etc.). All of this information is also represented as UnitData to be available for selection.

The table ParentUnit defines the containment hierarchy between units. A child unit inherits the category or corpus membership from its parent (see below). On creation of a base file, one representant unit is created that represents the whole basefile, such that the basefile categories are automatically propagated down to the children. Note that this relationship is in general not a tree, but a graph, since overlapping units that share sub-units are allowed. The possible containment hierarchy relations with the predefined unit types is shown in figure 14.6. Figure 17.1 shows an example of these note units on a sound file.

The table NextUnit records the logical order of units in a basefile. This speeds up the detection of a natural concatenation in the calculation of the concatenation distance. Note that one unit can have multiple next or previous units, e.g. one note unit has the following note unit and its attack sub-unit as next units.

14.2.4 Category Membership

Table IsIn records the membership of a unit to a category or corpus. We don't need to keep the transitive closure here, because this is already done in the IsA (A.1.3) hierarchy among categories, i.e. we can find out with one query all the categories a unit belongs to. This is also saving space since there are many more units than categories.

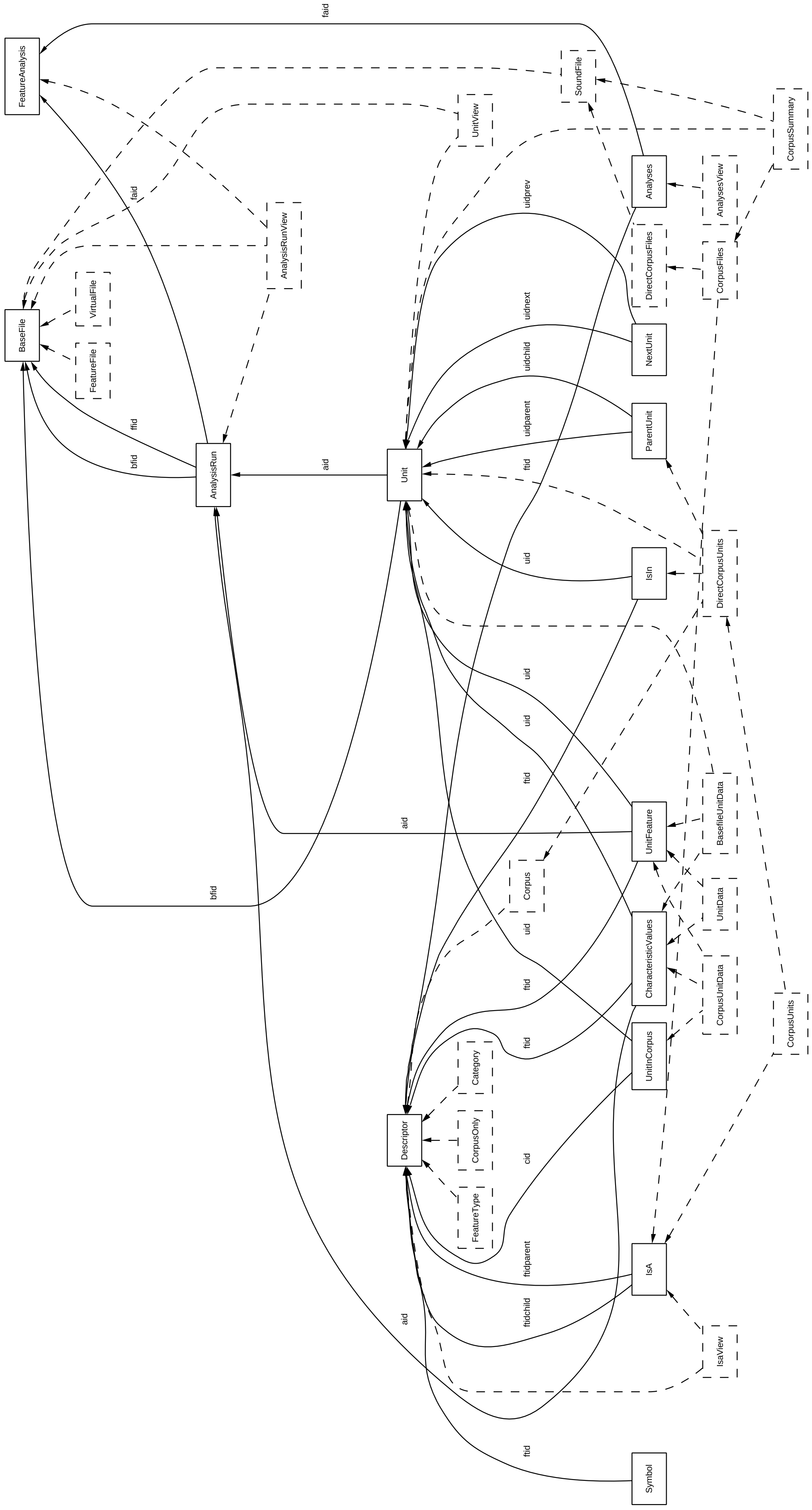


Figure 14.5: Full SQL implementation schema of the CATERPILLAR database in UML notation, showing tables with the foreign key references between them and views with the tables or views they depend on with dashed lines.

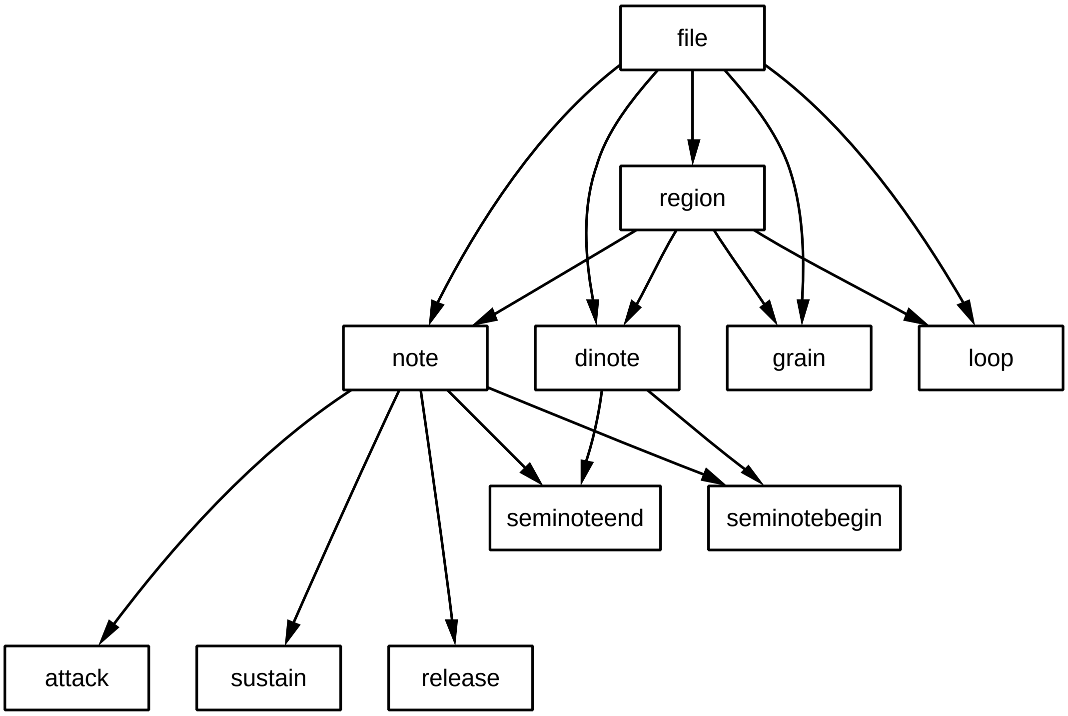


Figure 14.6: ParentUnit relationship for predefined unit types

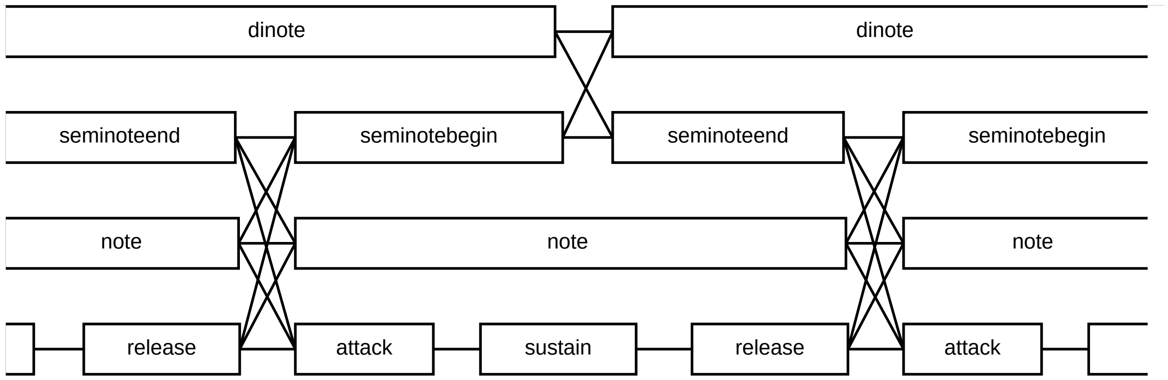


Figure 14.7: NextUnit relationship drawn as lines between notes, seminotes, dinotes, and subnote units.

Membership of a unit in a Category is expressed as a binary descriptor by the relationship IsIn. Because the database models a containment hierarchy parent of units, it is enough to add the highest parent unit (eventually the whole file which is represented as a unit, too, as will be explained in the next section) to a class. Because we also model an inheritance hierarchy IsA among classes, this adds all the contained units to the categories and all its base categories.

For example, we have a violin recording and have added the representant unit (see below) that encompasses the whole file to the category *violin*. All the note units are children of the file unit. When we determine what categories a unit is in, we query its categories as given by the IsIn relationship, and those of all parents. To the obtained categories, we add all the base categories of the latter, given by the IsA relationship. The result is that all units of the violin recording are also members of the category *strings* and *instrument*, because these are the base categories of *violin*, and all this by supplying just one explicit membership.

14.2.5 Representant Units

The CATERPILLAR database does not offer a membership relationship of sound files in categories or corpora. Although this is the most common case (after all, every sound file is part of several corpora, and most often in a category as a whole, e.g. for solo instrument recordings), it is also only a special case of category membership: we need to flexibly express categories for single units or regions in a sound file. In order not to double the mechanism for sound files and units, we express file membership as membership of a special *representant unit* that covers the whole file. This unit is the parent of all other units of this sound file. In order to add all the units of a sound file to a category, we then only need to add the representant unit to the IsIn relationship, as explained above.

Another use of representant units is as a container for corpus statistics: It is convenient and time-saving to calculate often used statistics of corpus data only once. These statistics are the mean value and standard deviation of a given descriptor and characteristic value. We could now define a table CorpusStatistics with the corpus id, the descriptor id, and each characteristic value as attributes, but this exactly the structure of the combined UnitData (A.4.3) and CharacteristicValues (A.4.2) tables. We can avoid this redundancy by creating a *corpus representant unit* for each corpus, that has two tuples of unit data per descriptor, for mean and standard deviation, two entries in AnalysisRun (A.2.2) (which also store the date of the last update of the statistics) with two different FeatureAnalysis (A.1.4) distinguishing the type of statistics.

All representant units are automatically created by trigger procedures called by the DBMS, so that we can always rely on their existence.

14.2.6 Descriptor Extraction

Table FeatureAnalysis describes an analysis program, and its parameters, called to compute raw descriptor data. We also store the name and the parameters of the analysis program used to perform an analysis of one or more FeatureTypes on a SoundFile, yielding one FeatureFiles.

Table Analyses records which feature types a feature analysis outputs. For future development, the system can call the analysis programs automatically, according these stored dependencies, like the Unix *make* utility.

It is indeed the common case that an analysis component hast multiple descriptor outputs, that are written all together to an SDIF file (section 13.4.1). To access the data of only one descriptor, Analyses stores the *sdif selection* to be appended to the feature file to retrieve just this descriptor.

The table AnalysisRun records that a base file was analysed by a feature analysis program producing a feature file. Its primary key aid is used in tables Unit, UnitFeature and CharacteristicValues to distinguish between different segmentations and the same descriptor data types produced by different analysis programs. Recognising the running of an analysis program as an entity avoids redundancies in the keys for unit data and characteristic values, and permits to record the date of the analysis in the database.

14.2.7 Data Tables

The working tables UnitFeature and CharacteristicValues contain the bulk of the data in the CATERPILLAR database: The characteristic values of the analysed descriptors of the units. The views UnitData, CorpusUnitData and BasefileUnitData perform the join of the two data tables, optimised for selecting all units from a corpus, or a basefile.

The table UnitFeature stores the mean value of a feature for a unit. It is complemented by table CharacteristicValues for non-constant features. We need to store both the analysis run id and the feature type, since one feature analysis can calculate several features, which can be in one single feature file.

This table complements the table UnitFeature with the characteristic values other than mean, discussed in chapter 11. It is only used to describe the continuous descriptors of each unit, i.e. static descriptors don't use space, and the necessary default values (see table 11.1) will be generated on-the-fly.

One essential point is the unification of features and categories as generic descriptors. The base class Descriptor has (conceptually) two derived classes implemented as the views Category and FeatureType. For the selection algorithm, and for displaying in the database explorer, the category membership is then interpreted as a boolean feature (normalised in the view UnitData). The category IDs become descriptor IDs.

14.3 Example Data and Queries

Some example queries show typical use cases of the Caterpillar database:

To get all units of a basefile (given by its id) of certain types are retrieved simply from table Unit (A.2.3) with:

```
SELECT uid, starttime, endtime, endtime - starttime AS duration, type
FROM   Unit
WHERE  bfid = basefile  AND  type IN (type-list);
```

To get all units in a corpus, we'd use a nested query to filter for the wanted *corpus-id*, and a union to catch the units and their parents.

```
SELECT uid
FROM   (SELECT uid, cid
        FROM   DirectCorpusUnits
       UNION
        SELECT uid, ftidparent AS cid
        FROM   IsA, DirectCorpusUnits
        WHERE  ftidchild = cid)
WHERE cid = corpus-id;
```

This query uses the view DirectCorpusUnits (A.3.2), which is implemented as follows:

```
CREATE VIEW DirectCorpusUnits AS (
  SELECT U.uid, C.ftid AS cid, U.type
  FROM   (SELECT uid, ftid              -- units directly in corpus
          FROM   IsIn
         UNION
          -- child units (transitive closure already done!)
          SELECT uidchild, ftid
          FROM   IsIn, ParentUnit
          WHERE  uidparent = uid        -- take their child units
         ) AS CU, Corpus C, Unit U
  WHERE CU.ftid = C.ftid  AND  CU.uid = U.uid
  AND   NOT C.isroot
  AND   U.type <> UnitType('corpus')  AND  U.type <> UnitType('file')
);
```

14.4 Future Extensions

The daily work with the Caterpillar database showed one case of use that was not satisfied by the current schema: For purposes of testing and visualisation, one often wants to limit the choice of units to one sound file. Now, this choice is given by a corpus, and most often there are many sound files constituting each corpus. We can achieve this choice by a preselection condition (see section 16.2), but a more general solution should be found. One possibility, considered for the design of CATERPILLAR, was to further unify the entities Corpus and SoundFile with the help of subclassing, i.e. to make BaseFile one more derived class (specialisation) of Corpus. This possibility was discarded since it would make the design unreadable, by obscuring fundamental differences between the entities, and creating spurious relations: Suddenly, a feature file would also be a derived class of descriptor. Furthermore, there are too few common attributes of the two (both have a name and can contain units) to warrant this unification. A better solution not yet implemented is to extend the idea of representant units (section 14.2.5) to representant corpus: Each time a basefile is added, a corresponding corpus is created, and all units will be part of this corpus. The latter is achieved by only adding the file representant unit to the file's corpus, since all other units are children of the file unit. Further advantages of this method is that by the corpus representant units, we get a place to store statistics of one sound file.

Another missing feature for specifying a corpus for synthesis is the possibility to subtract a sub-corpus from a larger corpus, e.g. to keep a validation set. This could be formulated as an intersection, like "take all units from corpus *Sonaten und Partiten* **but not** the first movement. Here, having a basefile represented also as a corpus is essential.

One performance bottleneck is the double tree expansion when accessing the data of all the child units of a unit in a category and all its base categories. It uses the view CorpusUnitData (A.4.4), which in turn, via 3 more views, joins 7 tables. (Please verify on figure 14.5.) It would be efficient to store all (given or implied) containment relationships in a vast table with judicious use of indices, which would be kept up-to-date by trigger procedures on insertion of new units or categories.

One impractical design detail is that in the transitive closure in the hierarchies IsA (A.1.3) and IsIn (A.2.4), an element is not its own parent. This would simplify the queries a lot, since at the moment, a union has to be made between the units directly in a corpus, and via inheritance. Having a self-recursive membership, we can do with just one query whose parents are in a certain category.

The CATERPILLAR database allows us to store references to all the sound and data files, the descriptor types and their analysis programs in FeatureAnalysis (A.1.4), the synthesis units and their descriptors. The analysis programs for all the desired features are called by the user. Table Analyses (A.1.5) adds to this the possibility to store the dependencies between the different analysis methods, such that the system knows by itself which programs to call to generate all the necessary input files for a descriptor one wants to calculate, similar to the Unix *make* tool.

We should also store the parameters of each selection in the database, along with the result, for documentation and comparison.

All the listed future extensions are easy to integrate because of the data independence. SQL allows us to change table definitions, add or remove columns, split a table into two and define a view that can be used exactly like the old table.

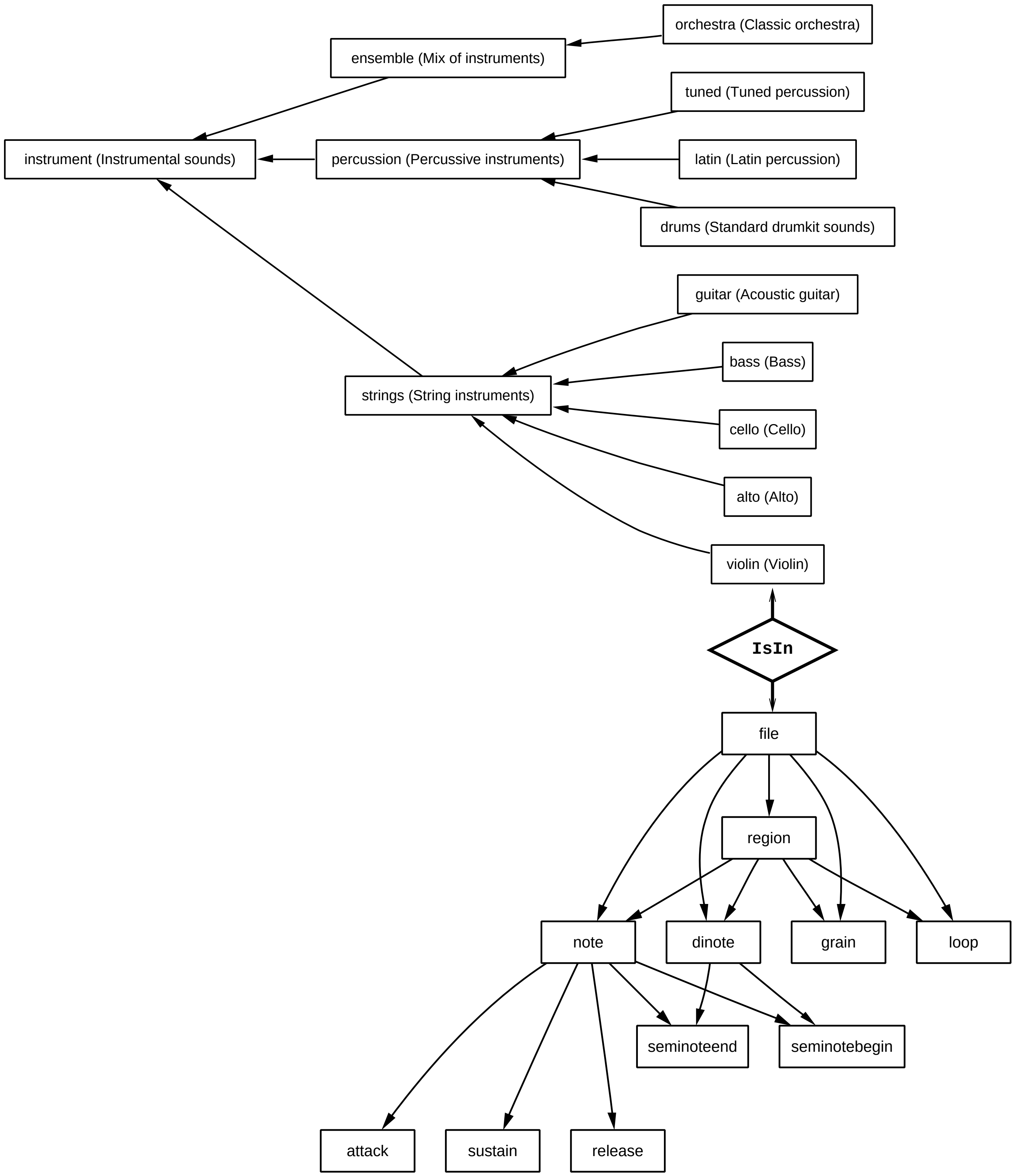


Figure 14.8: Membership of a unit and all child units in a category and all base categories.

Chapter 15

Corpus Examples and Statistics

This chapter gives examples of the actual content of the CATERPILLAR database, with statistics of the descriptor values. At the time of writing, the database contained 213 soundfiles of a total length of 3 hours and 17 minutes in 194812 units. Table 15.1 shows the number of units by unit type.

Type	Unit type	Num. Units
0	all	194812
1	file	217
3	note	25530
5	dinote	25486
6	attack	25518
7	sustain	13259
8	release	25518
13	corpus	343
15	seminotebegin	25545
16	seminoteend	25518
17	grain	27878

Table 15.1: Number of units in the CATERPILLAR database by unit type

All content is organised in corpora, which are grouped into a hierarchy shown in figures 15.2– 15.1. Remember that every category (see section 10.2) is also a corpus. To distinguish categories from corpora, the former are all lowercase, while the latter are capitalised.

In the following sections, we will show statistics of corpora containing violin pieces (section 15.1), voice (section 15.2), and environmental and effects sounds (section 15.3).

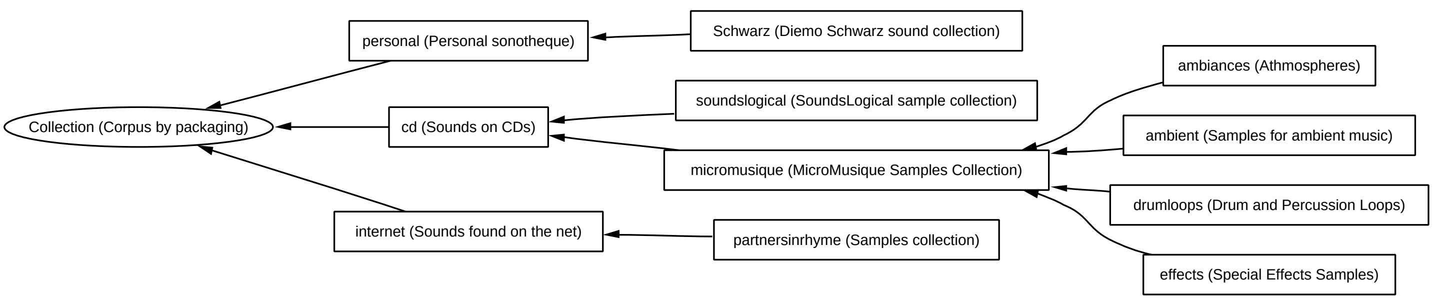


Figure 15.1: The corpus hierarchy for collections

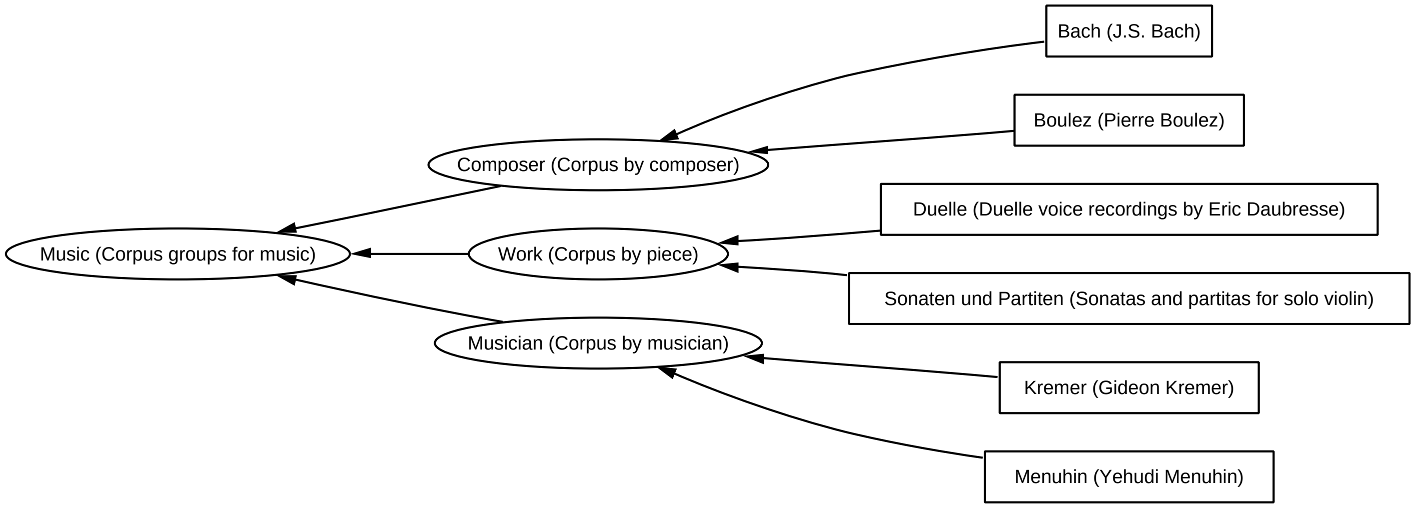


Figure 15.2: The corpus hierarchy for music

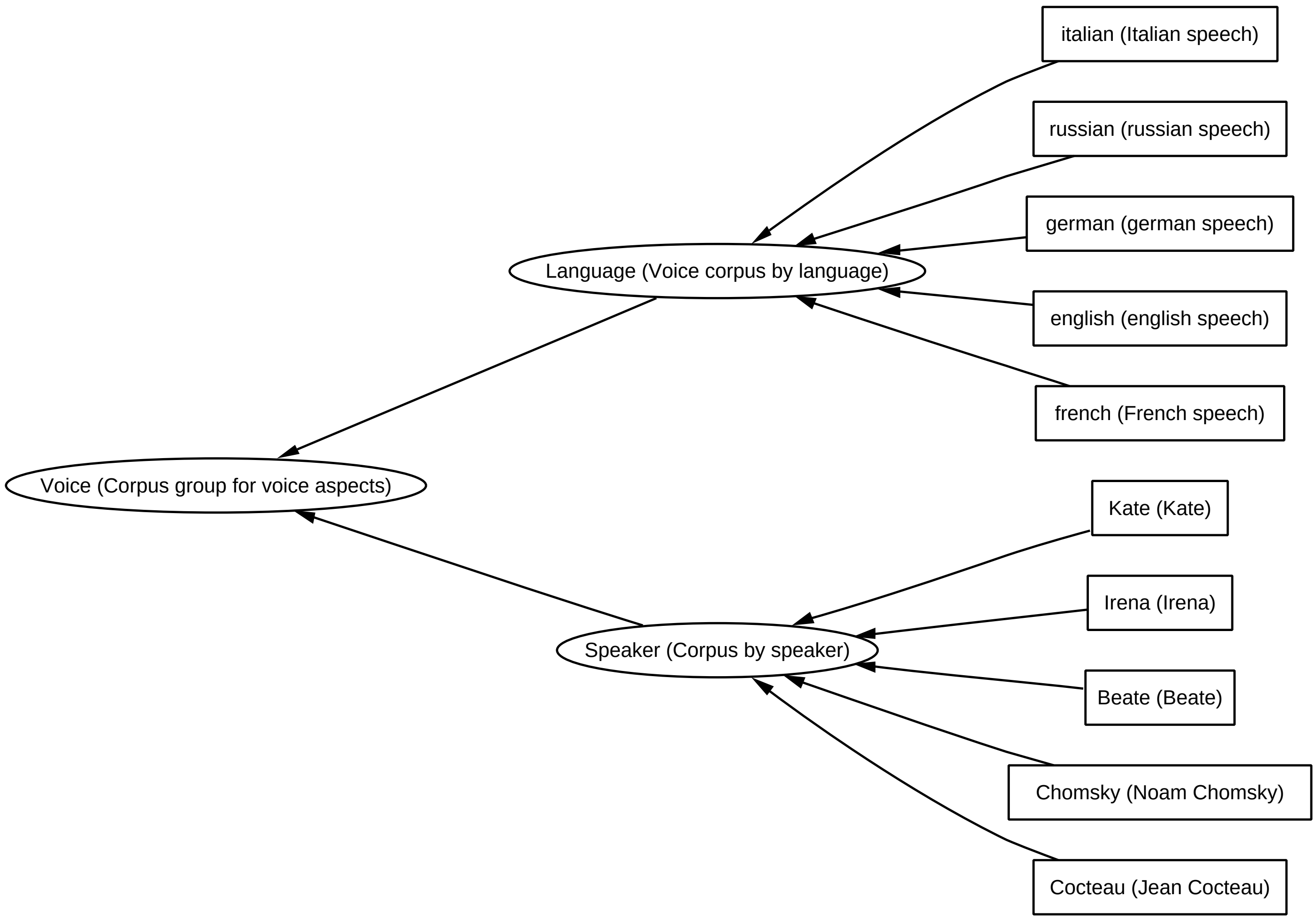


Figure 15.3: The corpus hierarchy for voice

15.1 Solo Violin Sonatas

For the instrument synthesis application (section 17.1), a corpus was built consisting of the *Sonata No. 1*, *2* and *3* for solo violin from J.S. Bach's *Sonaten und Partiten*, played by Yehudi Menuhin[1], and Gideon Kremer[2], each about 45 minutes long. See table 15.2 for the content of the violin corpus intended mainly for instrument synthesis, and its sub-corpora.

The sound files were segmented by aligning them with their Midi score files from the classical music archives[3] (Schwob 2003) by music alignment using DTW (section 6).

Figure 15.4 shows right away the problem of this corpus: a small part of units have an analysed mean fundamental frequency that does not correspond to the frequency given by the Midi note number. The majority of the notes lie on the correct correspondence that is visible as an exponential curve. The reason for these errors are shortcomings in the pitch analysis program and imprecisions in the alignment. The pitch errors are in part due to the 17% of polyphonic notes in the corpus (see figure 15.5).

Files	Units	Length	Corpus
21	114708	01:30:43	source / instrument
21	114708	01:30:43	source / instrument / strings
21	114708	01:30:43	source / instrument / strings / violin
21	114708	01:30:43	Music / Composer
21	114708	01:30:43	Music / Composer / Bach
20	114511	01:30:29	Music / Musician
10	57210	00:43:13	Music / Musician / Kremer
10	57301	00:47:16	Music / Musician / Menuhin
20	114511	01:30:29	Music / Work
20	114511	01:30:29	Music / Work / Sonaten und Partiten
8	45004	00:31:40	Music / Work / Sonaten und Partiten / Sonata 1
8	60536	00:43:27	Music / Work / Sonaten und Partiten / Sonata 2
4	8971	00:15:22	Music / Work / Sonaten und Partiten / Sonata 3

Table 15.2: Content of violin category and corpus for instrument synthesis and sub-corpora

[1] CD Johann Sebastian Bach, *Sonates et Partitas pour violon seul*, recorded 1956–1957, published 1993 by EMI
[2] CD Johann Sebastian Bach, *The Sonatas and Partitas for Solo Violin*, recorded 1980, published 1981 by Philips
[3] `http://www.classicalarchives.com`

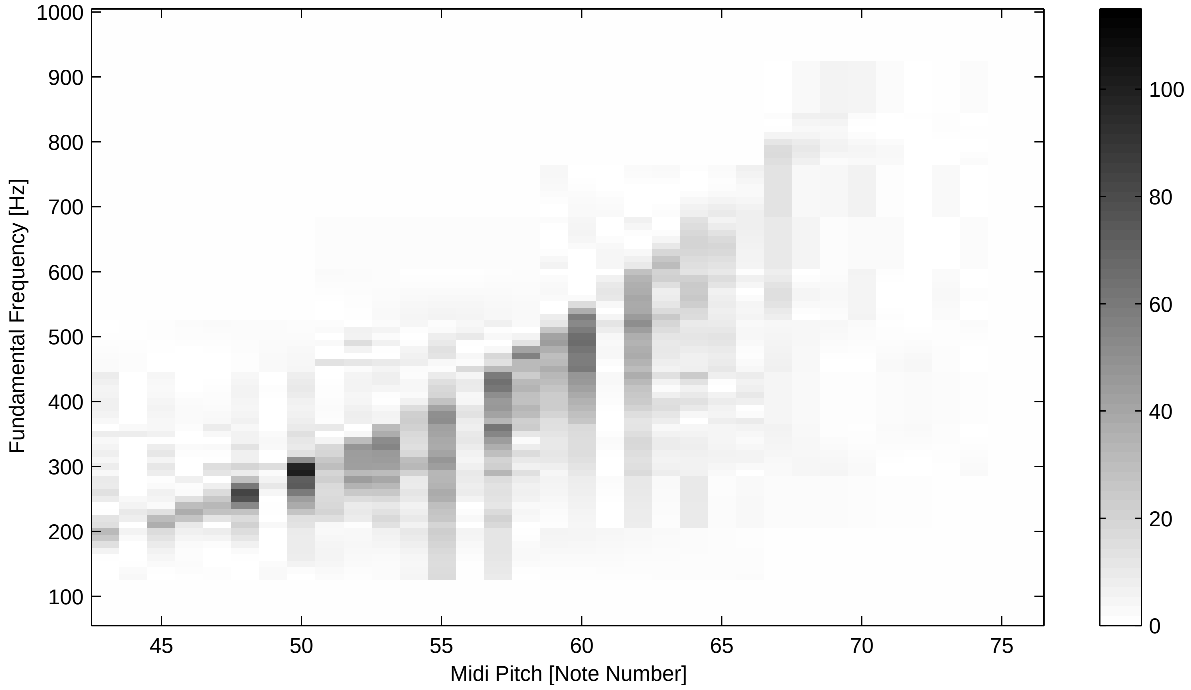
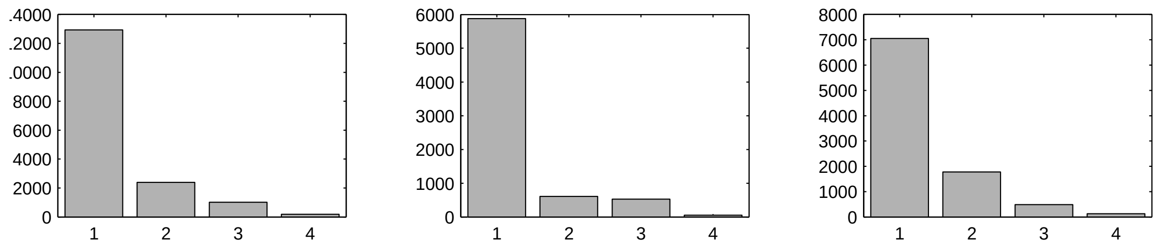
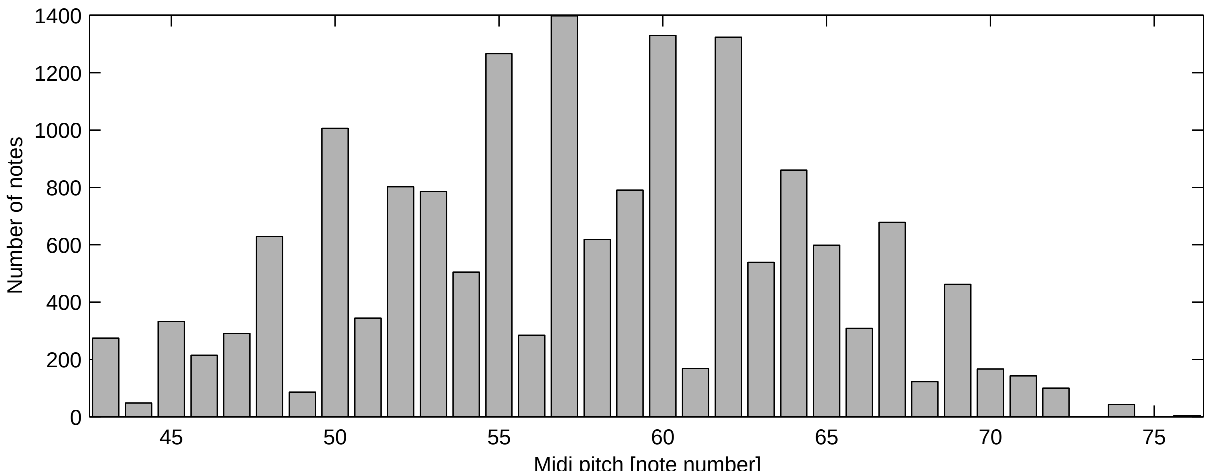


Figure 15.4: Histogram of fundamental frequency over Midi pitch for note units of corpus *Sonaten und Partiten*

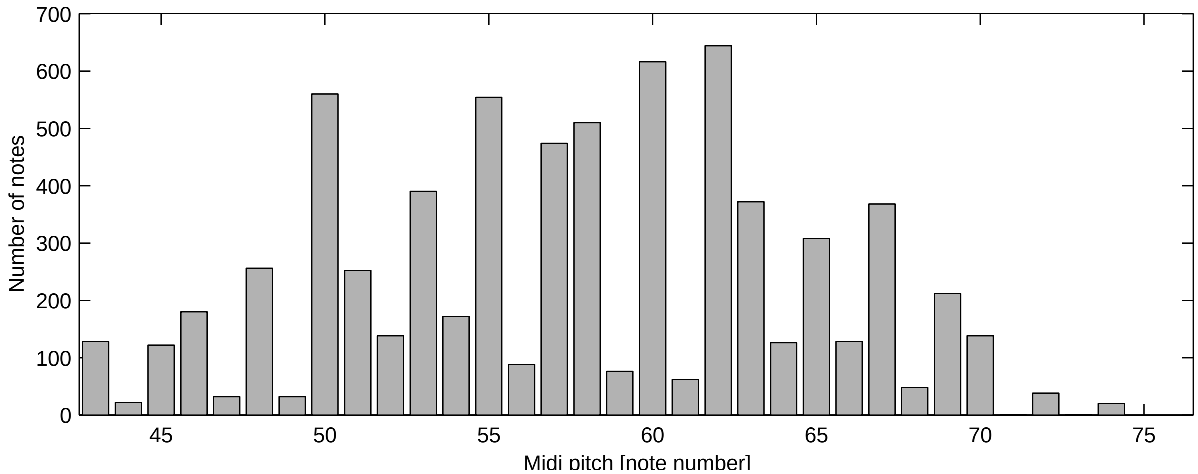


(a) *Sonaten und Partiten* (b) *Sonata No. 1 in G minor* (c) *Sonata No. 2 in A minor*

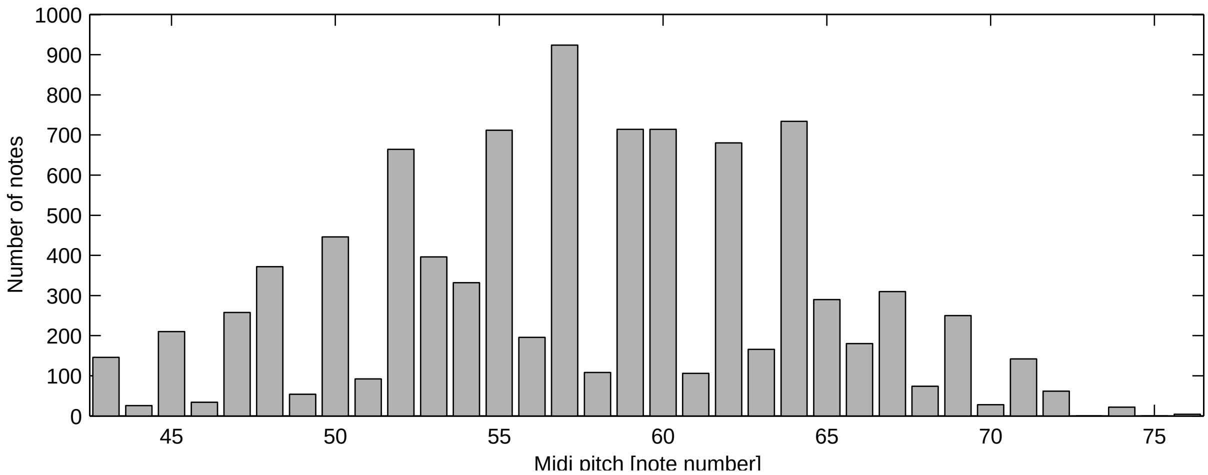
Figure 15.5: Histogram of Midi polyphony of violin corpora



(a) *Sonaten und Partiten*

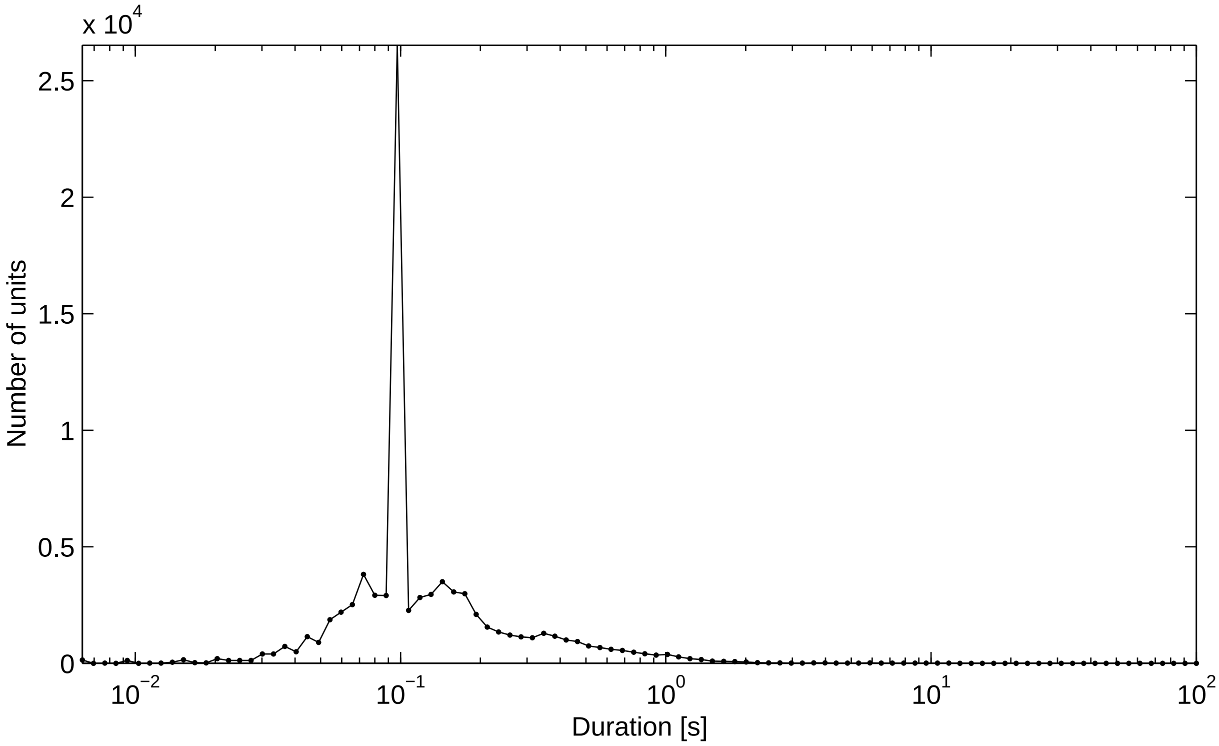


(b) *Sonata No. 1 in G minor*

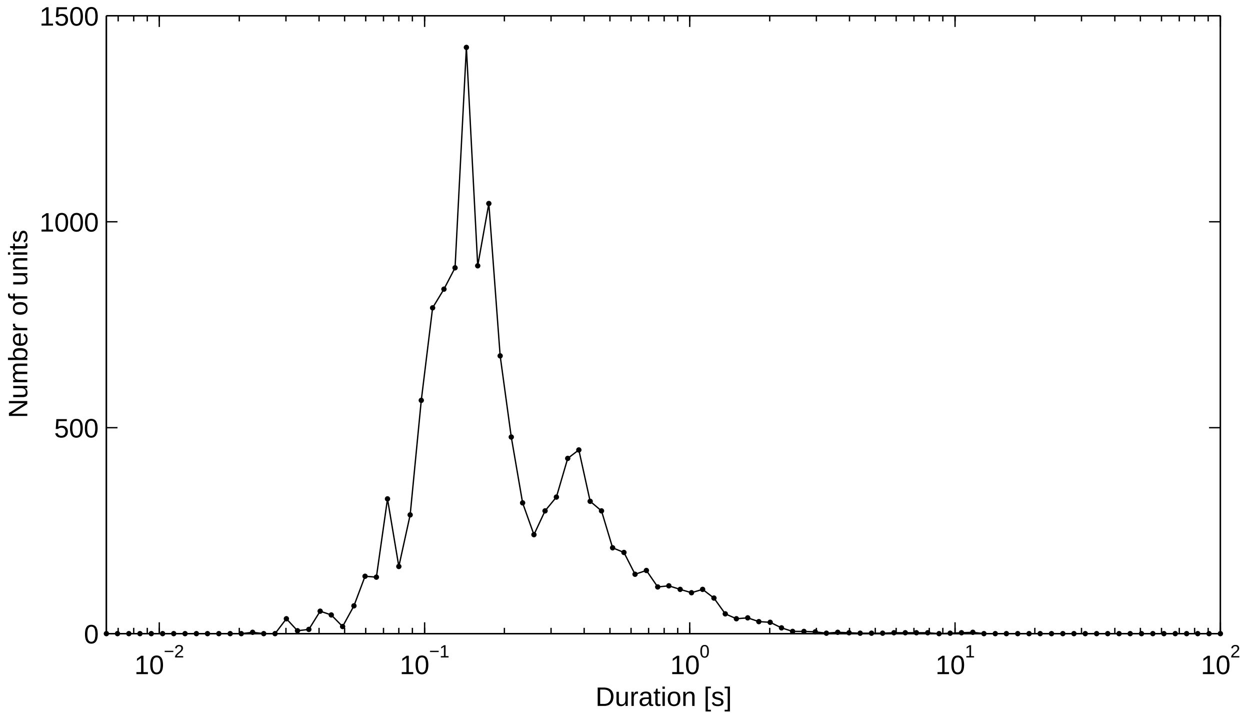


(c) *Sonata No. 2 in A minor*

Figure 15.6: Histogram of Midi note number of violin corpora

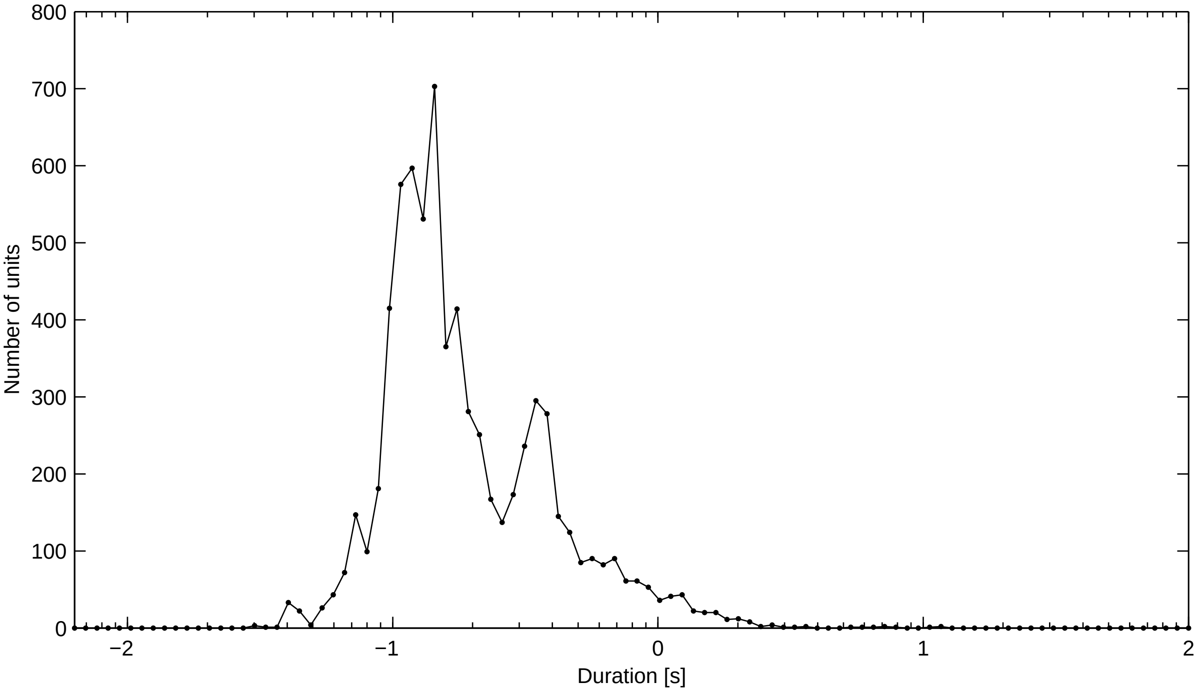


(a) All units

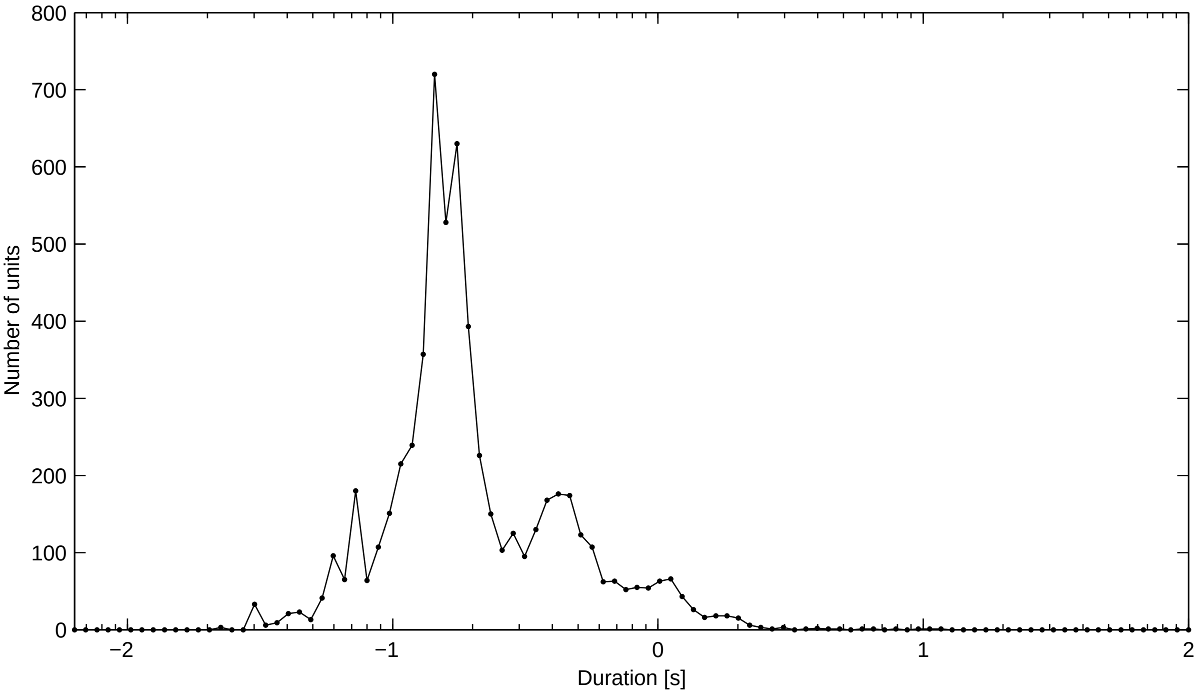


(b) Only note units

Figure 15.7: Histogram of duration of corpus *Sonaten und Partiten*. The spike at 100 ms in (a) is due to the sub-segmented attack and release units fixed to that length (see section 17.1.1).

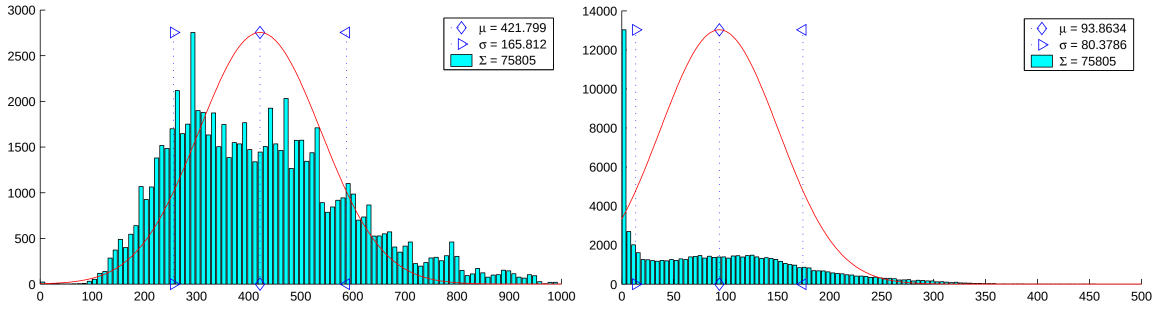


(a) *Sonata No. 1 in G minor*

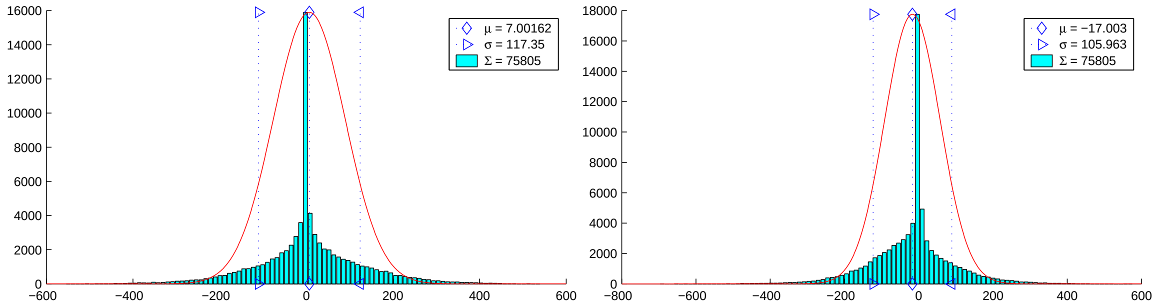


(b) *Sonata No. 2 in A minor*

Figure 15.8: Histogram of duration of note units of corpora *Sonata 1* and *Sonata 2*

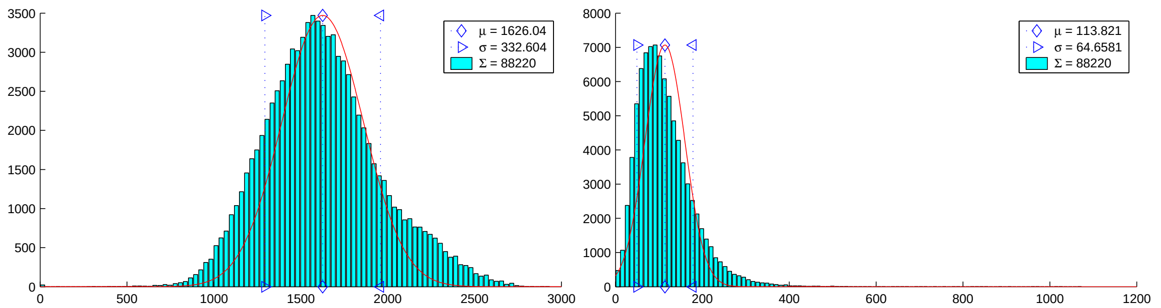


(a) mean (b) standard deviation

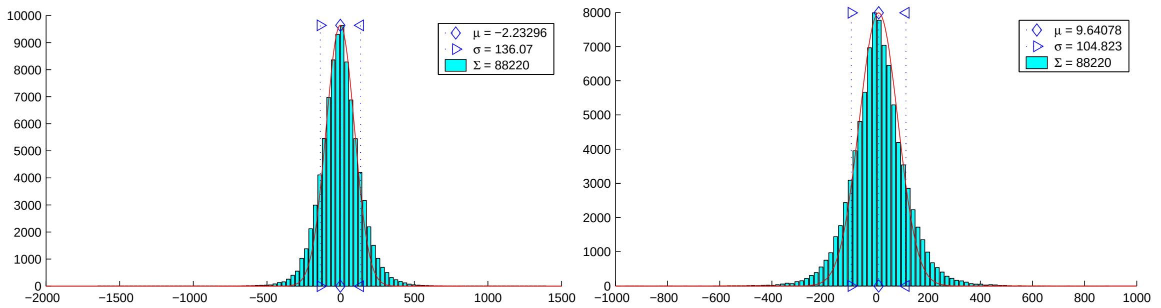


(c) slope (d) curve

Figure 15.9: Histogram of pitch value characteristics for corpus *violin*

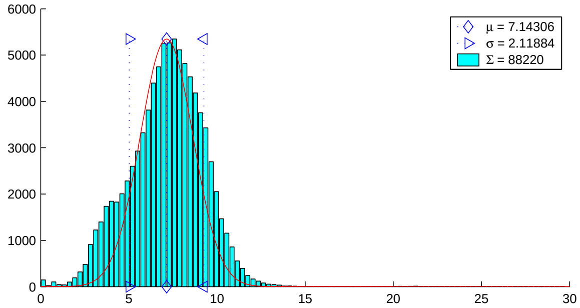


(a) mean (b) standard deviation

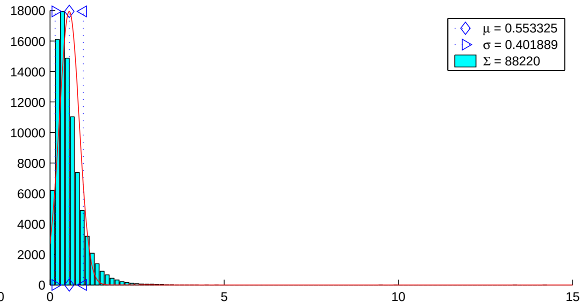


(c) slope (d) curve

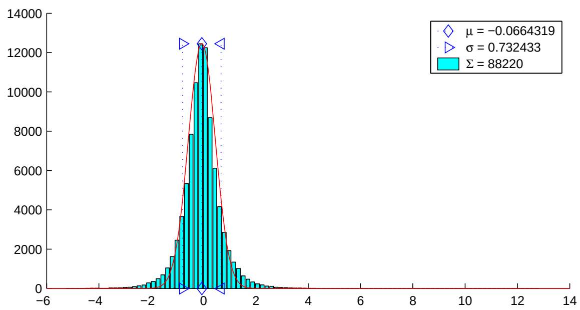
Figure 15.10: Histogram of spectral centroid value characteristics for corpus *violin*



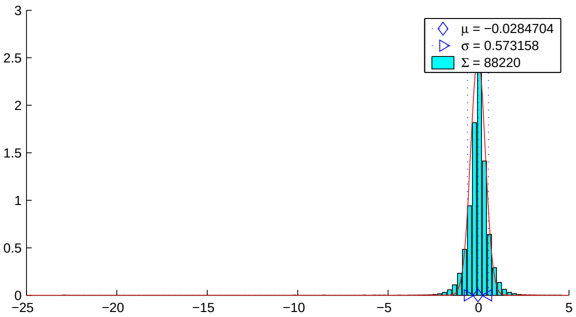
(a) mean



(b) standard deviation

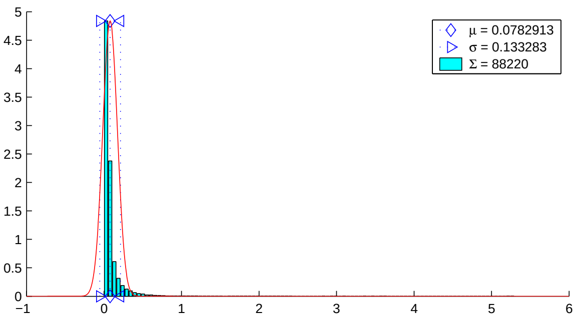


(c) slope

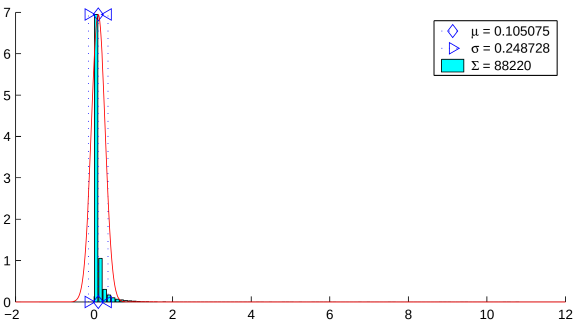


(d) curve

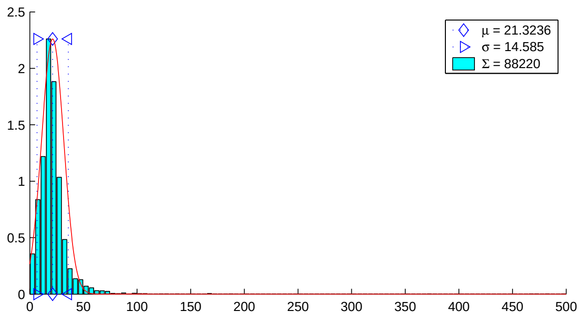
Figure 15.11: Histogram of loudness value characteristics for corpus *violin*



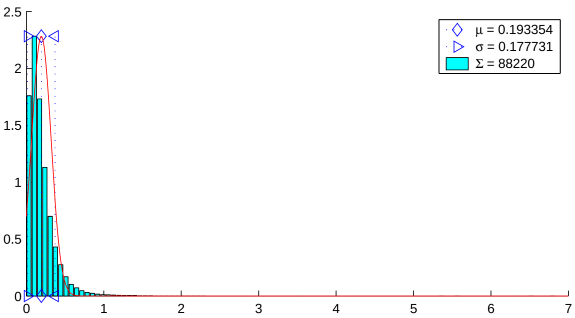
(a) AR attack time



(b) AR release time

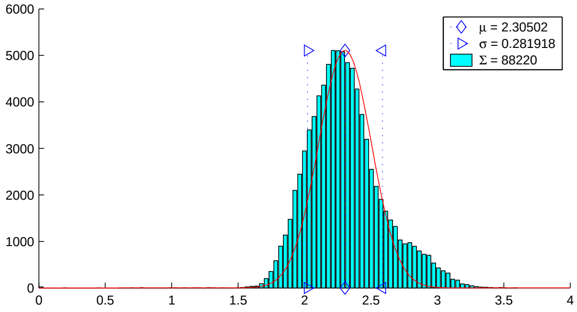


(c) descriptor spectrum centroid

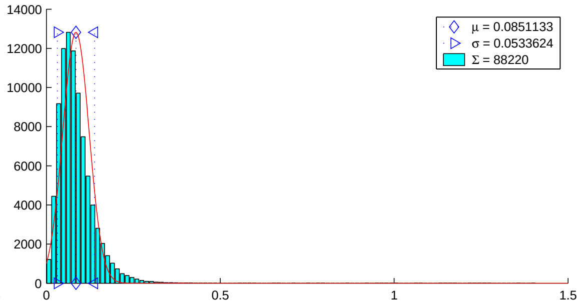


(d) polynomial residual

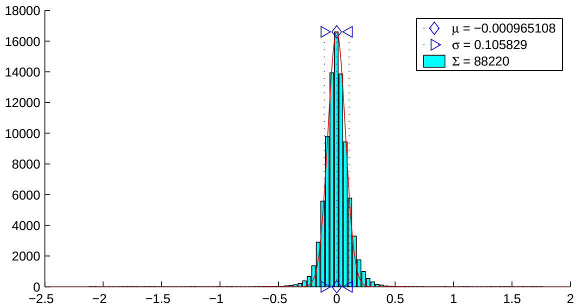
Figure 15.12: Histogram of loudness spectrum characteristics for corpus *violin*



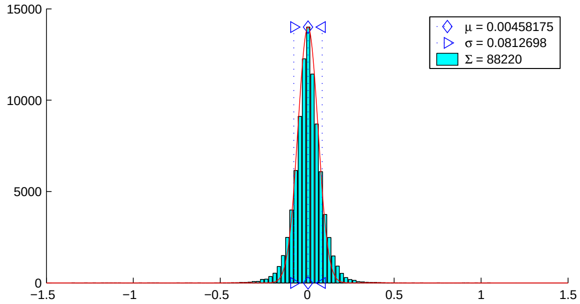
(a) mean



(b) standard deviation

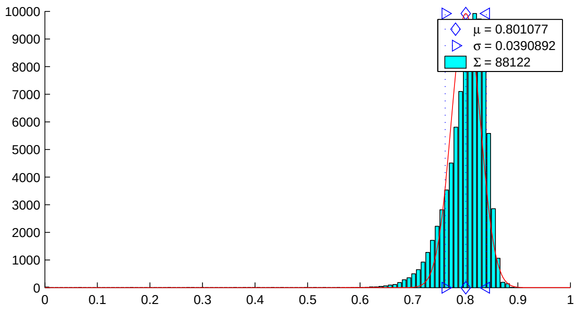


(c) slope

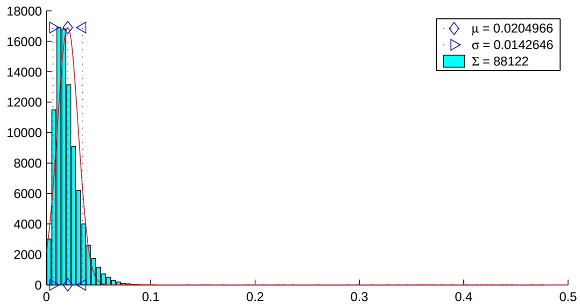


(d) curve

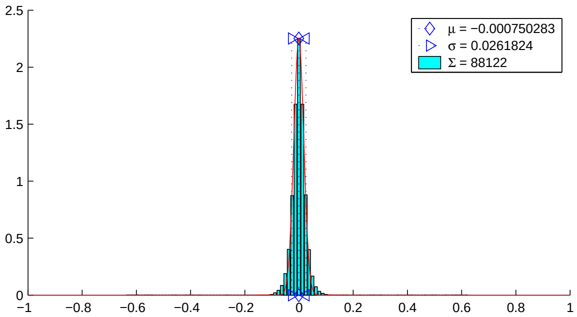
Figure 15.13: Histogram of spectral sharpness value characteristics for corpus *violin*



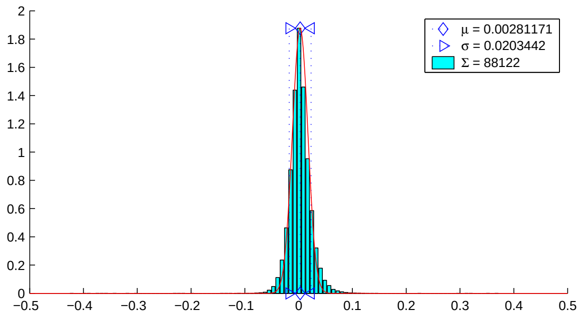
(a) mean



(b) standard deviation



(c) slope



(d) curve

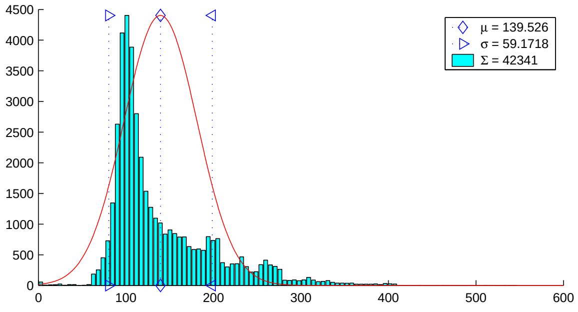
Figure 15.14: Histogram of timbral width value characteristics for corpus *violin*

15.2 Voice

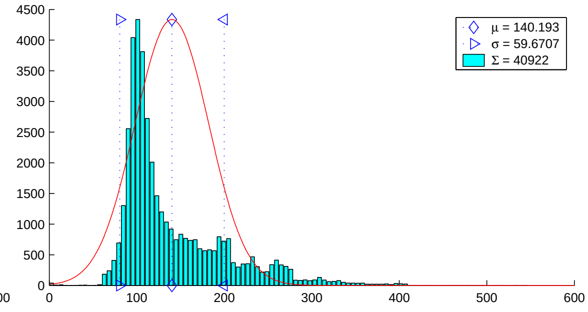
The voice corpus in CATERPILLAR is segmented into phones by a quite erroneous blind segmentation method (Rossignol 2000). Therefore, it is not suitable for speech synthesis, but allows to play with voice snippets from various speakers and in various languages in the free synthesis application (section 17.4).

Files	Units	Length	Corpus
33	43326	00:45:55	source / voice
33	43326	00:45:55	source / voice / human
18	1437	00:01:03	source / voice / human / female
15	41889	00:44:52	source / voice / human / male
25	9826	00:07:55	Voice / Language
1	723	00:00:36	Voice / Language / english
18	1437	00:01:03	Voice / Language / french
2	2188	00:01:50	Voice / Language / german
4	5478	00:04:26	Voice / Language / russian
22	50278	00:51:44	Voice / Speaker
2	2188	00:01:50	Voice / Speaker / Beate
6	13287	00:18:51	Voice / Speaker / Chomsky
4	5478	00:04:26	Voice / Speaker / Irena
1	723	00:00:36	Voice / Speaker / Kate
9	28602	00:26:01	Voice / Speaker / Shafqat

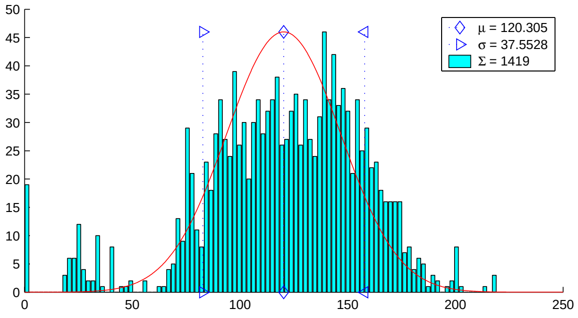
Table 15.3: Content of voice categories and corpora



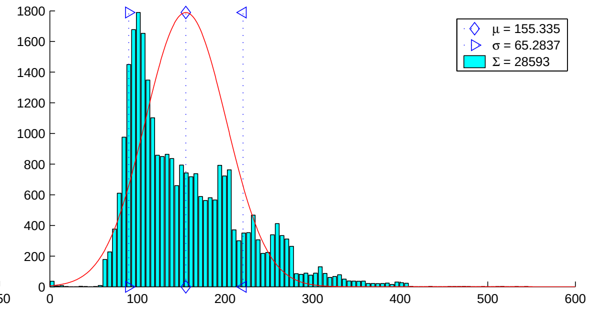
(a) Mean f0 for voice corpus



(b) Mean f0 for male voice corpus

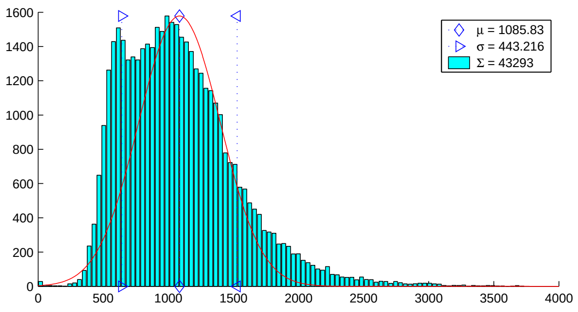


(c) Mean f0 for female voice corpus

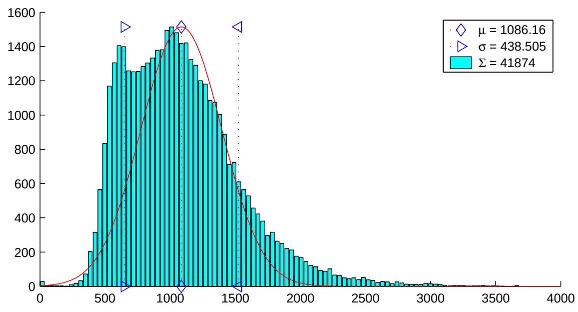


(d) Mean f0 for Shafqat Ali Khan corpus

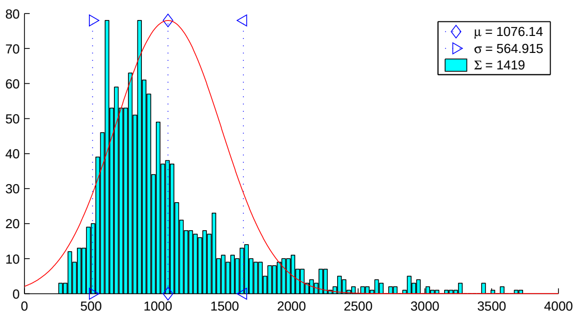
Figure 15.15: Histograms for voice corpora



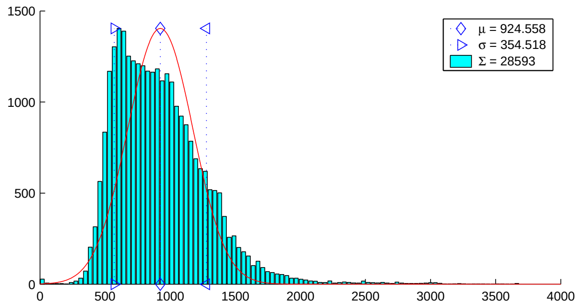
(a) Mean centroid for voice corpus



(b) Mean centroid for male voice corpus

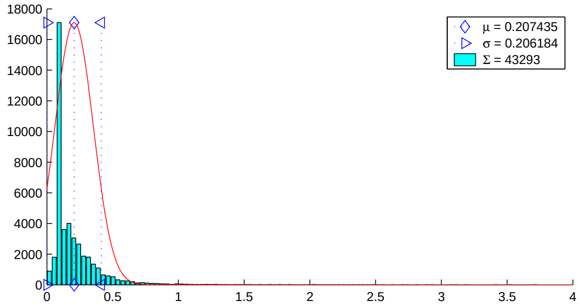


(c) Mean centroid for female voice corpus

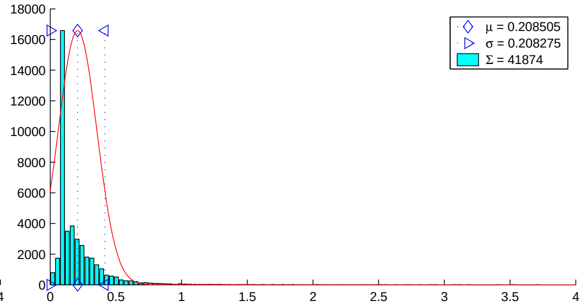


(d) Mean centroid for Shafqat Ali Khan corpus

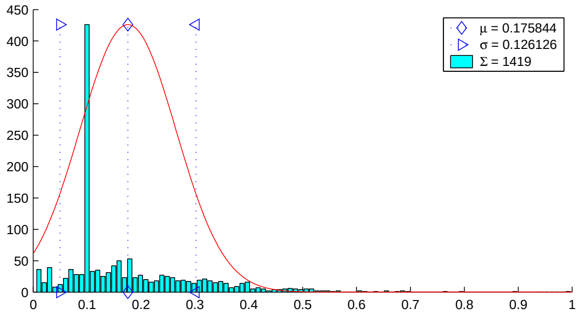
Figure 15.16: Histograms for voice corpora



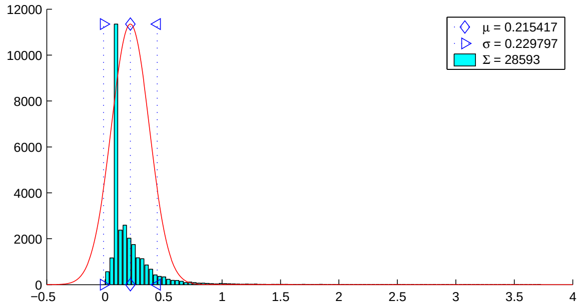
(a) Duration for voice corpus



(b) Duration for male voice corpus



(c) Duration for female voice corpus



(d) Duration for Shafqat Ali Khan corpus

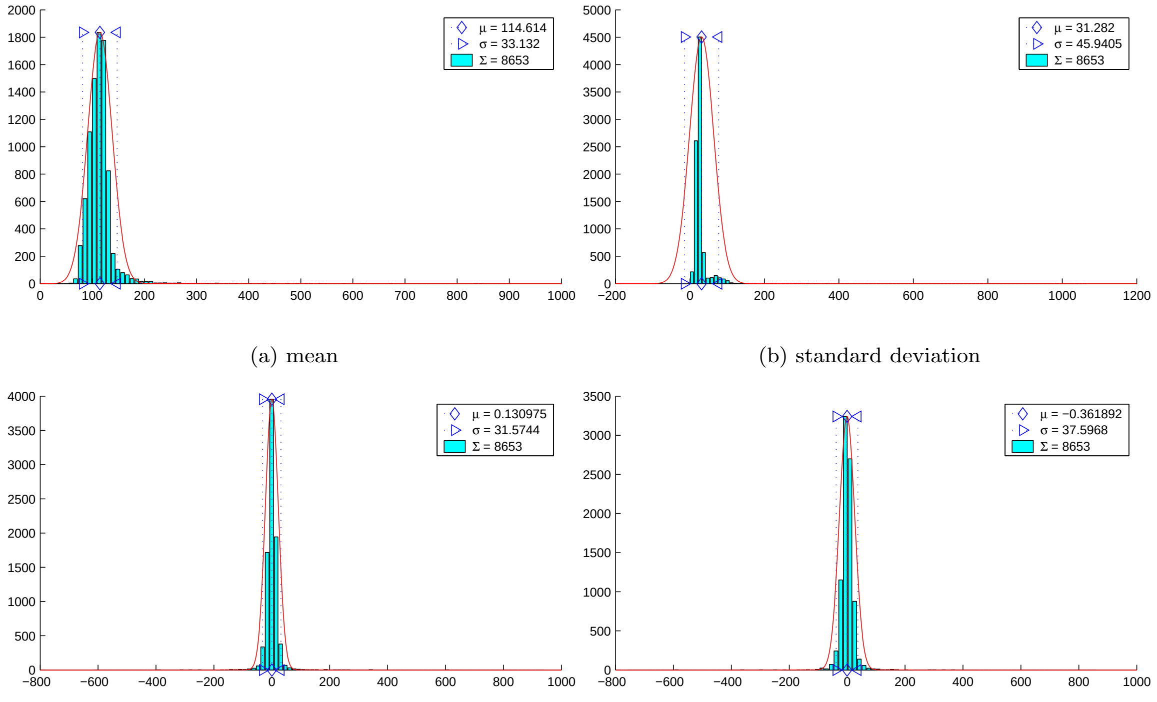
Figure 15.17: Histograms for voice corpora

15.3 Environmental and Effects Sounds

The various noise sounds have been segmented by chopping them into grains of fixed lengths of 0.5 and 0.225 s. They serve as material in the resynthesis and free synthesis applications (sections 17.2 and 17.4).

Files	Units	Length	Corpus
17	10929	00:28:11	source / noise
17	10929	00:28:11	source / noise / environment
2	47	00:00:07	source / noise / environment / elements
2	47	00:00:07	source / noise / environment / elements / water
4	83	00:00:12	source / noise / environment / people
4	83	00:00:12	source / noise / environment / people / crowd
11	10799	00:27:52	source / noise / environment / traffic
6	2233	00:05:45	source / noise / environment / traffic / car
5	8566	00:22:08	source / noise / environment / traffic / train
26	39531	00:54:11	Collection
15	30835	00:31:45	Collection / cd
9	28602	00:26:01	Collection / cd / Shafq-o-matic
132	16623	00:25:13	Collection / cd / micromusique
26	5440	00:07:47	Collection / cd / micromusique / ambiances
79	8815	00:12:38	Collection / cd / micromusique / ambient
13	507	00:00:42	Collection / cd / micromusique / drumloops
14	1861	00:04:05	Collection / cd / micromusique / effects
6	2233	00:05:45	Collection / cd / soundslogical
6	130	00:00:18	Collection / internet
6	130	00:00:18	Collection / internet / partnersinrhyme
5	8566	00:22:08	Collection / personal
5	8566	00:22:08	Collection / personal / Schwarz

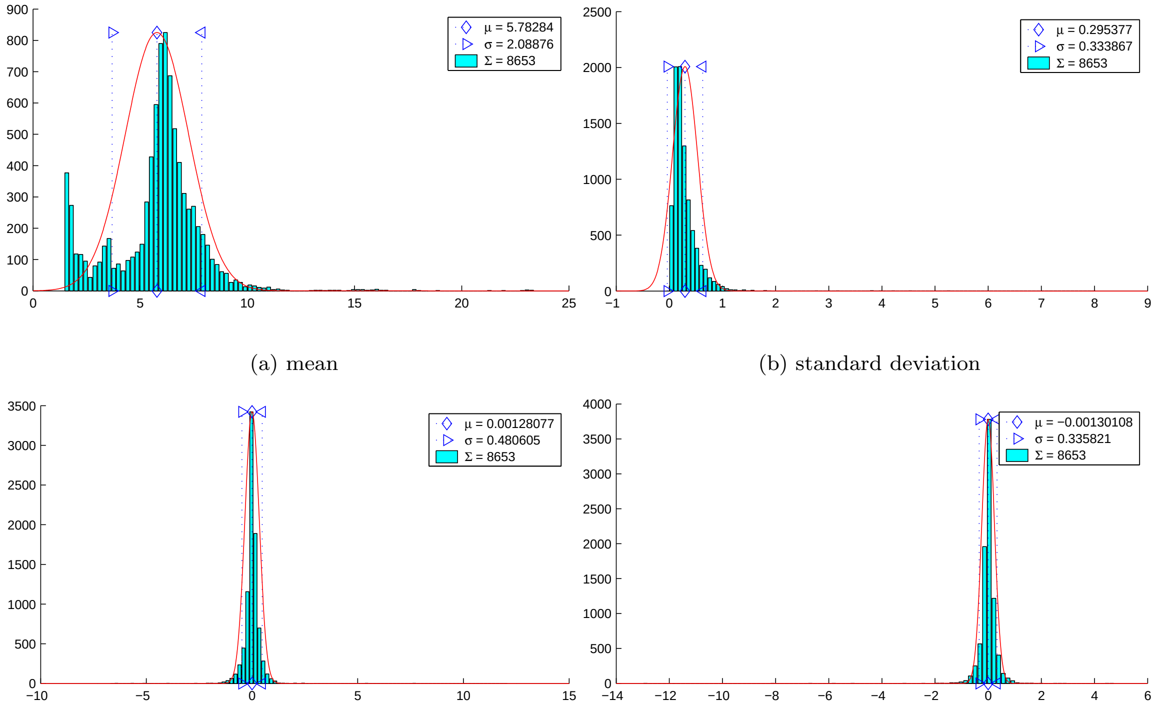
Table 15.4: Content of environmental and effects sound categories and corpora



(a) mean (b) standard deviation

(c) slope (d) curve

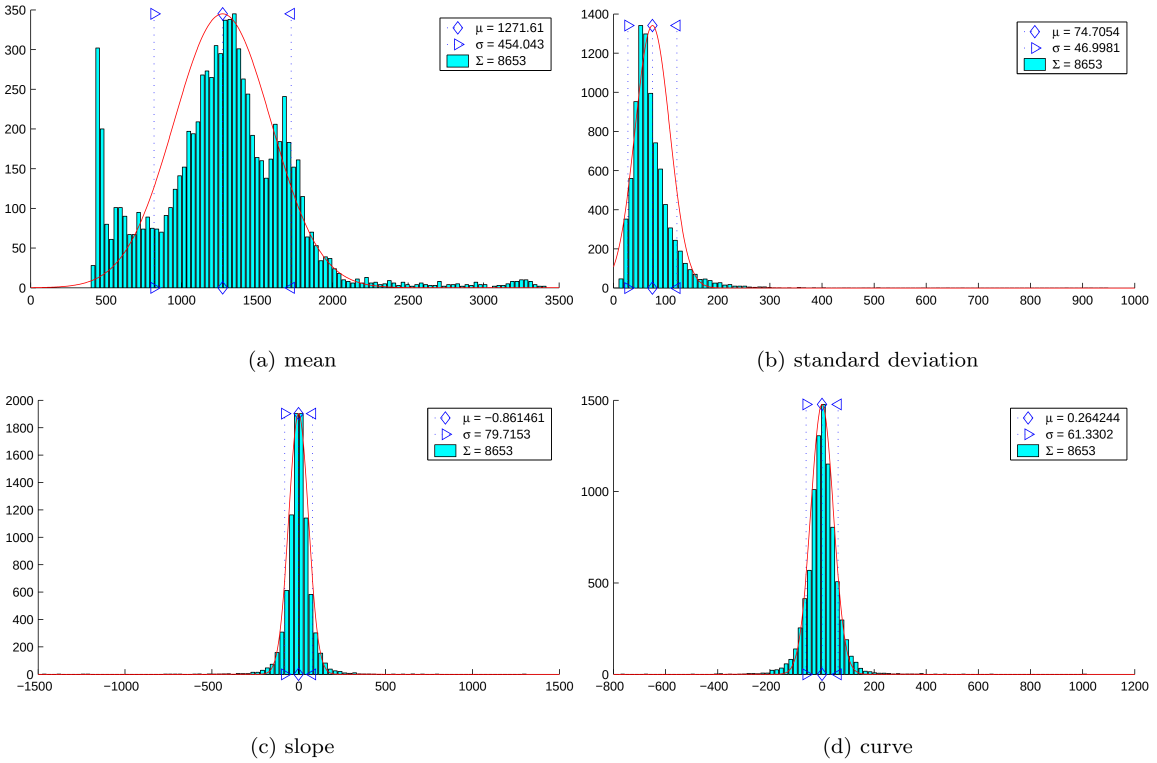
Figure 15.18: Histogram of pitch value characteristics for corpus *environment*



(a) mean (b) standard deviation

(c) slope (d) curve

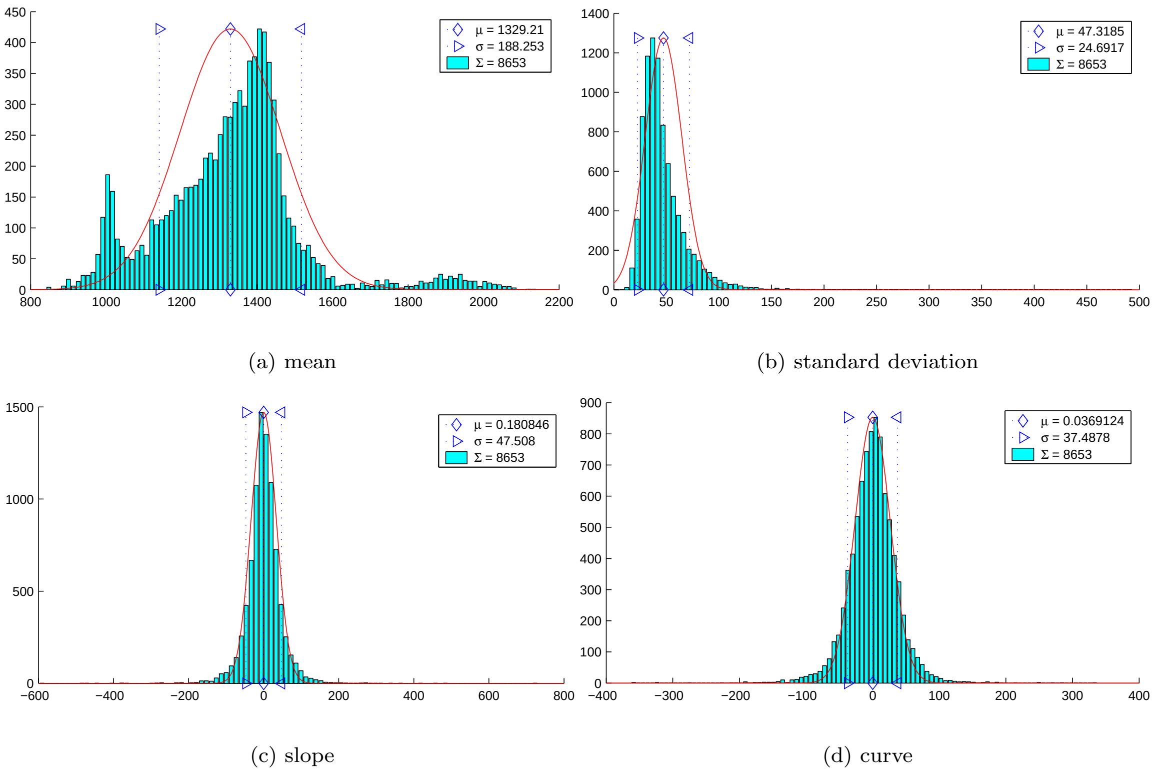
Figure 15.19: Histogram of loudness value characteristics for corpus *environment*



(a) mean (b) standard deviation

(c) slope (d) curve

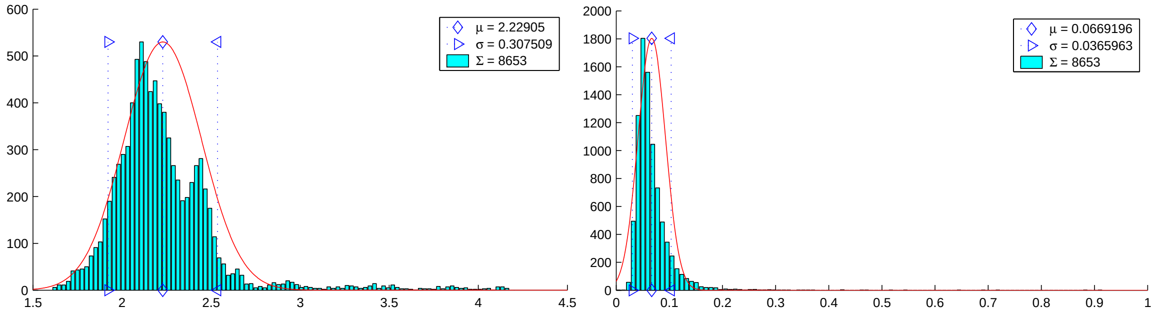
Figure 15.20: Histogram of spectral centroid value characteristics for corpus *environment*



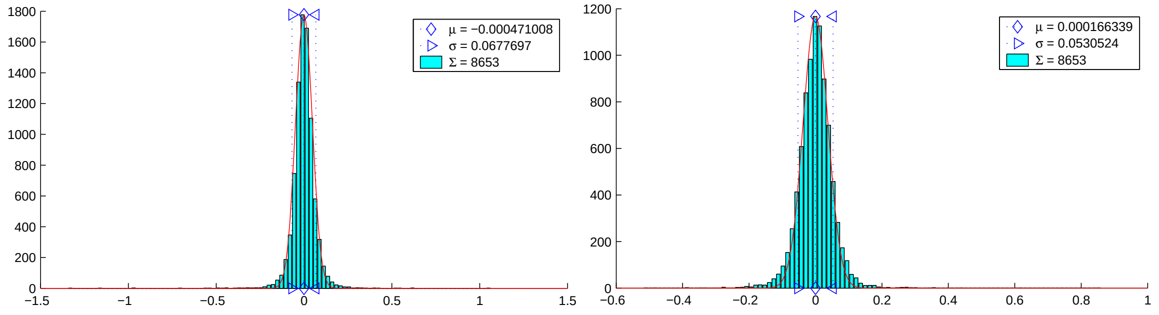
(a) mean (b) standard deviation

(c) slope (d) curve

Figure 15.21: Histogram of spectral spread value characteristics for corpus *environment*

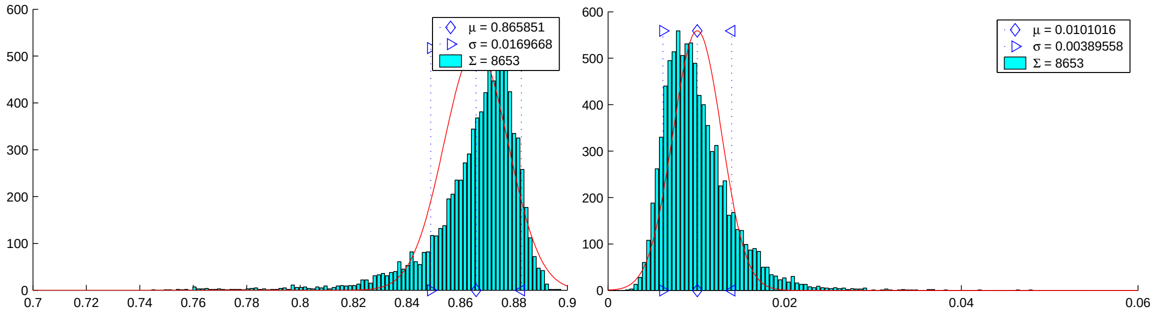


(a) mean (b) standard deviation

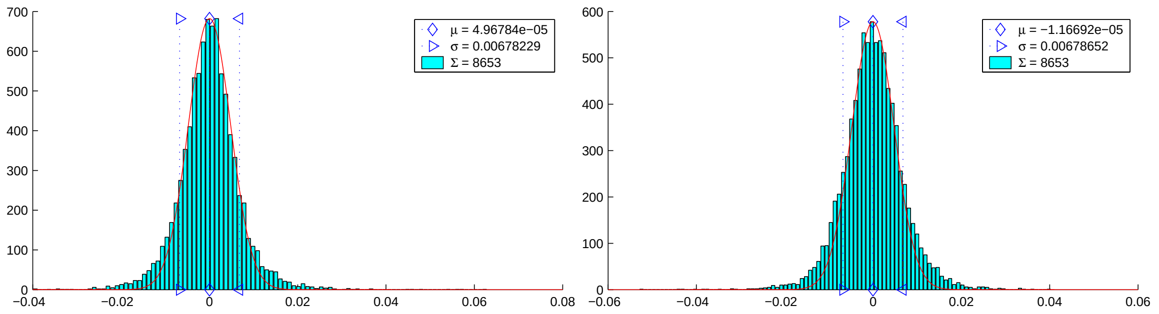


(c) slope (d) curve

Figure 15.22: Histogram of spectral sharpness value characteristics for corpus *environment*



(a) mean (b) standard deviation



(c) slope (d) curve

Figure 15.23: Histogram of timbral width value characteristics for corpus *environment*

Chapter 16

Synthesis

Synthesis in CATERPILLAR is done by the four steps preselection, unit selection, possibly transformation, and concatenation. The selection of the units from the database is done by a unit selection algorithm that finds the units that best match the given synthesis target. This match is defined by a number of *distance functions*, described in section 16.1, that predict the acoustic similarity between a database unit and a target unit, or the quality of the concatenation between two database units.

Before selection, we have to choose which units in the database are appropriate for our synthesis target. This *preselection* is described in section 16.2.

The *unit selection* algorithm is crucial as it contains all the "intelligence" of data-driven concatenative synthesis. Two different algorithms are described in the following: a path-search algorithm (section 16.3) which is an extension of the classic algorithm used in speech synthesis, and an algorithm based on a constraint solving approach (section 16.4).

The selected units are possibly transformed to better match the target, and concatenated. These steps of *transformation* and *concatenation* are described in section 16.5.

16.1 Distance Functions

The distance functions are the basis for the costs that guide the unit selection algorithm to find the optimal sequence of units to satisfy the target. According to the type of the descriptor (dynamic, static, or categorical — see chapter 10), we have to apply different sorts of distance functions.

16.1.1 Distances for Dynamic or Static Descriptors

16.1.1.1 Euclidean Distance

Each characteristic value (see chapter 11) of a dynamic descriptor can be used to calculate a distance. Static descriptors have reasonable default values for all characteristic values, listed in section 11.5. Most of the time, however, we will use the mean value of the descriptor.

The generic distance measure is a Euclidean distance on the characteristic values x normalised over the corpus, in order to avoid distortions between different distances because of the different ranges of the values.

The normalised value x' is given by

$$x' = \frac{x - \mu}{\sigma} \tag{16.1}$$

with μ the mean and σ the standard deviation of this characteristic value over the corpus.

The Euclidean distance d for a database unit's value x'_u and a target value x'_t is then

$$d = \sqrt{(x'_u - x'_t)^2} \tag{16.2}$$

In practice, to find the units with the smallest distance, we can use the square of this distance to avoid the costly square-root computation.

16.1.1.2 Special Distances

Some special distances exist that replace the term $x'_u - x'_t$ in the above equation: The *duration distance* is asymmetrical and returns a high distance when a database unit is shorter than a target unit, because this would make a hole in the selected sequence.[1] We could return infinity, but this would be too strict. Having a unit a few milliseconds shorter than required might be the better choice, so we prefer to penalise shorter units by returning the exponential of the difference value (starting from 1). If the database unit is longer than the target, the distance is the time difference. The rationale is that we can always shorten a unit, but it is best to select a unit with a similar duration, for instance to capture the articulation of a recorded note for instrument synthesis.

16.1.2 Distances for Category Descriptors

Category descriptors express either the membership of a unit in a class from a class hierarchy, such as *sound source*, or a value from a list of discrete symbols, e.g. *unit type*. We can not use the Euclidean distance from above, since a category descriptor does not have a numerical representation as such.

To make units nevertheless comparable in the domain of such a descriptor, we must define special distance functions in one of the following ways:

16.1.2.1 Boolean Distance

The simplest way is to use the membership of the two units to compare and to set the distance to zero if both are in the same class, and to infinite (or some high value) if they are in different classes. This is used in CATERPILLAR for example for the *unit type* descriptor, when we want to synthesise from dinotes, or an attack unit, etc.

16.1.2.2 Distance Matrix

The most generally applicable solution is to specify a distance matrix explicitly, as for example for the phoneme class for speech synthesis. An example can be found in (Prudon 2003). An expert specifies a distance value for every possible pair of n categories to completely fill the n^2 elements of the matrix. This is only feasible for a small number of categories.

16.1.2.3 Data-Driven Distance

It is possible to define a data-driven category distance by performing a statistics of a subset of dynamic descriptors on each category. The distances between the category mean values in the descriptor space define then the distance between categories. The problem here is the choice of descriptors to use.

16.1.2.4 Tree Distance

We can derive a *tree distance* between two categories A and B automatically from the database hierarchy of categories, embodied by the IsA (A.1.3) relationship. Remember that this specialisation hierarchy is stored with its transitive closure, and that the number of levels between two nodes is stored with each edge. We can thus query the lowest common ancestor category C of A and B, and can directly access the respective tree distances d_{AC} and d_{BC} (the path lengths in the graph). To

[1]We do not perform a stretching transformation on a database unit. However, for voice synthesis, this would be quite necessary and could be feasibly done using PSOLA techniques (section 3.1.2.4).

define the tree distance between A and B, d_{AB} we have the choice of using either $\min(d_{AC}, d_{BC})$, the minimum, or $\max(d_{AC}, d_{BC})$, the maximum number of nodes to reach C, or to use the sum $d_{AC} + d_{BC}$, which is the length of the path between A and B. However, this tree distance is not used in CATERPILLAR because it has the major flaw that these distances depend too much on the level of detail of the hierarchy: A part of the tree where intermediate categories were introduced will suddenly be more distant from another part of the tree, without there being an intrinsic reason for this.

16.1.2.5 Similarity Distance

An improvement and extension of the tree distance is proposed by Pachet, Roy, and Cazaly (2000), where they use a taxonomy, i.e. an undirected graph of similarity relationships, to define distances between categories.

The CATERPILLAR category tree is in fact a directed acyclic graph, because each category can have multiple parent categories in table IsA (A.1.3). By adding special “similarity” edges to that table (that can carry a value for the degree of similarity), we could represent the category distance measure, without breaking its use for the classification. (Remember that the true parent edges serve to express class membership, as described in section 12.2.2.)

This possibility is not exploited in CATERPILLAR, since we did not yet encounter the need for expressing transversal links. Moreover, defining the similarity measure by specifying the links is a nontrivial task. Pachet, Roy, and Cazaly (2000) describe a taxonomy for musical styles that embodies a global knowledge about music, that has been constructed by an expert. Data-mining in user’s data like playlists or peer-to-peer networks might unveil relationships, but it is hard to know which.

16.2 Preselection

Before unit selection can be performed, a choice has to be made which database units to consider. The first choice, of course, is that of a corpus to use. Then, often only a certain type of unit within a corpus is to be used, e.g. only note units. Further, units with extreme characteristics have to be filtered out, e.g. very short units, or units with very low energy. And finally, units that we know are not useful for our synthesis target can be eliminated, for instance polyphonic units of usually monophonic instruments such as the violin, when we only want to synthesise a monophonic phrase. Here is also the place to weed out imperfect dinotes due to errors in the segmentation by alignment: The pitch transition width described in section 11.2 gives a good indicator if we have a proper dinote, when the width is smaller than 10 milliseconds.

All these *a priori* choices are grouped into the stage of *preselection*. Preselection is performed before units are loaded into memory and directly on the level of the database, i.e. the unit criteria are translated into conditions for the SQL query that loads the unit data into memory for selection.

Furthermore, for some applications, a preselection per target unit would be applicable, which corresponds to a fixed and known class we want to use for a specific target unit, such as the phoneme class for speech synthesis (see section 17.5). Care has to be taken, however, to not limit the selection too much. It can be preferable to select a unit from a different but close class than a bad matching unit from the given class.

Preselection is referred to as *Sound Sieve* in the content-based retrieval and data-driven synthesis system MOSIEVIUS (Lazier and Cook 2003), see also section 4.3.3.

Interestingly, the recent work by Blouin and Bagshaw (2003) and Blouin (2003) identifies and formalises the necessity for preselection also for speech synthesis and presents methods to automatically determine the best set of units from which to perform the actual unit selection. However, Blouin defines three phases using a different terminology: First, the *choice of corpus*, which corresponds to our preselection. Second, the *preselection* that determines for each target unit a small set of units (20–40) that match best the target. This phase corresponds to our pruning (see section 16.3.4). Third, the actual selection calculates *only* the concatenation distances, presuming that all preselected units match well.

16.3 The Path Search Unit Selection Algorithm

The unit selection algorithm used in CATERPILLAR is based on the standard path search algorithm used in speech synthesis described in section 16.3, first proposed in (Hunt and Black 1996). However, as there are some specifics, it is described again here in detail.

The path search unit selection algorithm finds the sequence of database units u_i that best match the given synthesis target units t_τ using two cost functions: The *target cost* expresses the similarity of u_i to t_τ including a context of r units around the target. The *concatenation cost* predicts the quality of the concatenation of u_i with a preceding unit u'. The optimal sequence of units is found by a Viterbi algorithm finding the best path through the network of database units.

In the following, we describe the details of the path search unit selection algorithm. The algorithm finds the units from the database of N units u_i that best match the T given synthesis target units t_τ. The quality of the match is determined by two costs, given by weighting an application-dependent subset of the distance functions described in section 16.1:

16.3.1 Target Cost

The *target cost* C^tcorresponds to the perceptual similarity of the database unit u_i to the target unit t_τ. It is given as a sum of p weighted individual feature distance functions C_k^t as:

$$C^t(u_i, t_\tau) = \sum_{k=1}^{p} w_k^t \, C_k^t(u_i, t_\tau) \tag{16.3}$$

To favour the selection of units out of the same context in the database as in the target, the *context cost* C^xor *extended target cost*, for the sake of the mnemonic, considers a sliding context in a range of r units around the current unit with weights w_j decreasing with distance j.

$$C^x(u_i, t_\tau) = \sum_{j=-r}^{r} w_j^x \, C^t(u_{i+j}, t_{\tau+j}) \tag{16.4}$$

The context cost is a generalisation of the descriptors explicitly modeling context used in concatenative speech synthesis, such as the phonetic context. Our approach of a general context cost that can be applied to all features is, because of its greater flexibility, better suited to a creative musical use of concatenative synthesis.

16.3.2 Concatenation Cost

The *concatenation cost* C^cexpresses the discontinuity introduced by concatenating the units u_i and u_j from the database. It is given by a weighted sum of q feature concatenation cost functions C_k^c:

$$C^c(u_i, u_j) = \sum_{k=1}^{q} w_k^c \, C_k^c\,(u_i, u_j) \tag{16.5}$$

The cost depends on the unit type: concatenating an attack unit allows discontinuities in pitch and energy, a sustain unit does not. Consecutive units in the database (the pairs that are in relationship NextUnit in figure 14.1) have a concatenation cost of zero. Thus, if a whole phrase matching the target is present in the database, it will be selected in its entirety, leading to nonuniform unit selection.

We use, for instance, the distance between pitch end value of the left and pitch start value of right unit plus the difference in slope, for the concatenation of two units in their sustain phase. For pitch, we can take the descriptor's FFT spectrum into account to match similar vibrato frequency and intensity.

16.3.3 Finding the Optimal Unit Sequence

The unit database can be seen as a fully connected state transition network through which the unit selection algorithm has to find the least costly path that constitutes the target. Using the weighted extended target cost $w^t C^x$ as the *state occupancy cost* b_{ij}, and the weighted concatenation cost $w^c C^c$ as the *transition cost* a_{ij}, the optimal path can be efficiently found by a Viterbi algorithm (Viterbi 1967; Forney 1973). For new each target unit t_τ, we extend the array of shortest paths ψ from each candidate unit by one column τ, keeping only the minimal cost path continuation (as a back-pointer, i.e. the row index of the previous path element).

for $1 \leq j \leq N$:
 $a_{j1} = C^x(t_1, u_j)$
for $2 \leq \tau \leq T$:
 for $1 \leq j \leq N$:
  $b_{j\tau} = w^t C^x\,(t_\tau, u_j)$
  $a_{j\tau} = \min\limits_{1 \leq k \leq N} (a_{k,\tau-1} + w^c C^c\,(u_k, u_j) + b_{j\tau})$
  $\psi_{j\tau} = k_{\min}$

The decoding of the path ψ to find the optimal sequence of units s_τ is done in reverse order by first finding the path endpoint k with the least global cost a_{kT}, and then following the backward indices:

$k = \operatorname*{argmin}\limits_{1 \leq j \leq N}(a_{jT})$
for $T \geq \tau \geq 1$:
 $s_\tau = u_k$
 $k \;= \psi_{\tau k}$

16.3.4 Search Path Pruning

The asymptotic computational complexity of the unoptimised Viterbi algorithm is $O(TN)$ calculations of the target cost $C^x$and $O(TN^2)$ calculations of the concatenation cost C^c. This shows us that we need to reduce the number of possible concatenation edges in the state transition network. This is achieved, similar to Black and Campbell (1995), by *pruning* the network by only keeping the W best candidate units for each target. This reduces the total complexity to $O(TN + TW^2)$. In our tests, a rather large value for W of 500 was used in order not to limit the choice of units, regarding the corpus of 80000 units.

16.4 Unit Selection by Constraint Solving

As we have seen in the previous section, unit selection can be formalised as a best path search in a network. However, for musical applications, other conditions must be met for esthetically satisfying results. For example, synthesising a constant target sequence, we would not want the result to be constituted by the same unit repeated over and over, however closely that unit matches the target features. We want to synthesise a sequence of varying units *close enough* to the target, and avoid repetition. Or, as composing sounds is generally an iterative process, after an initial synthesis we'd want to refine the result, lock some units in place, and throw out some unpleasing units. All of this can be achieved under the formalism of *constraint satisfaction*. It has been first proposed for music program generation by Pachet, Roy, and Cazaly (2000) (see section 4.5) and for data-driven concatenative musical synthesis by Zils and Pachet (2001) in the *Musical Mosaicing* system described in section 4.3.1.

In this section, we propose a reformulation of the unit selection algorithm as a constraint satisfaction problem (CSP) and show examples how additional constraints can be specified. We use the *adaptive local search* algorithm described in detail in (Codognet and Diaz 2001; Truchet, Assayag, and Codognet 2001).

Constraint satisfaction is defined by an error function, which allows us to easily express the unit selection algorithm as a CSP using the target and concatenation costs between units. Indeed, one can argue that path-search unit selection is a special case of adaptive local search unit selection where each target unit is visited only once.

Constraints are given by distances, which allows to easily express the target and concatenation costs between units as constraints.

A problem in CSP form is given by:

The *variables* V_i, for which we seek a configuration that satisfies our constraints, are the sequence of unit indices to select. The *global cost function* to minimise is the total selection cost:

$$\begin{aligned} C^s = {} & w^t C^x \left(u_{V_1}, t_1\right) + \\ & \sum_{i=2}^{T} w^c C^c \left(u_{V_{i-1}}, u_{V_i}\right) + w^t C^x \left(u_{V_i}, t_i\right) \end{aligned} \tag{16.6}$$

Each *unit constraint* $\mathbf{C}_i$ comprises the variables V_{i-1} through V_{i+1}. The error function for one constraint is given by the target cost for the unit u_{V_i}, and the concatenation cost to the adjacent units $u_{V_{i-1}}$ and $u_{V_{i+1}}$ in the current configuration.

$$\begin{aligned} E(\mathbf{C}_i) = {} & w^t C^x \left(u_{V_i}, t_i\right) + \\ & w^c C^c \left(u_{V_{i-1}}, u_{V_i}\right) + w^c C^c \left(u_{V_i}, u_{V_{i+1}}\right) \end{aligned} \tag{16.7}$$

The adaptive local search algorithm starts from a random configuration of variables and repeats the following steps until the global cost drops under a given threshold, or a maximal number of iterations is reached:

1. For each constraint, compute its error and distribute it over the variables appearing on the constraint. A variable V_i is responsible for its target error $w^t C^x \left(u_{V_i}, t_i\right)$ and half of each concatenation error $w^c C^c \left(u_{V_{i-1}}, u_{V_i}\right)$, $w^c C^c \left(u_{V_i}, u_{V_{i+1}}\right)$, which are summed.
2. The unit in the variable with the highest error not marked as taboo is replaced by the best matching unit u_j with
$$j = \underset{1 \le j \le N}{\operatorname{argmin}} \left(\begin{array}{l} w^t C^x \left(u_{V_i}, t_i\right) + \\ w^c C^c \left(u_{V_{i-1}}, u_{V_i}\right) + \\ w^c C^c \left(u_{V_i}, u_{V_{i+1}}\right) \end{array} \right) \tag{16.8}$$
3. If no unit with lower error exists, mark the variable as taboo for a given number of iterations.
4. If all variables are marked taboo, restart with a new random configuration.

Additional constraints are:

- The *all different* constraint says that no unit must appear twice in the selection. It distributes a high penalty over the variables containing repeated units.
- *Unit ban* adds a high penalty to the variables containing a banned unit.
- *Unit forcing* lowers the target error of variables containing the units to include at the desired position.
- *Unit lock* marks the variable with the unit to lock as taboo.

To summarise, with CSP unit selection we sacrifice the real-time synthesis oriented efficient[2] linear selection of units in the path-search unit selection for a more composer-oriented interactive choice approach. This adds computational complexity, but also adds flexibility and interactivity.

[2]The asymptotical complexity of the dynamic programming path-search unit selection algorithm is $O(TN^2)$.

16.5 Transformation and Concatenation

The last stage of synthesis is the transformation and concatenation of the selected units. Transformation changes the selected database units to match more closely the target. Concatenation is concerned with providing a transition between the transformed units with minimal audible artefacts. This can mean performing transformation of the edges of the units. In short, transformation tries to reduce the target distance, and concatenation the concatenation distance on the already selected units.

16.5.1 Transformation

Transformation can change the loudness, pitch, duration and spectral content of a unit. Depending on the degree of change applied to a unit, the sound quality can suffer degradation. Also, for pitch and duration transformation, other sound representations than the sampled signal are of great advantage. The most applied to music are the additive representation (see section 10.7 and Risset and Mathews 1969; Serra and Smith 1990; Rodet 1997a), and PSOLA (see section 3.1.2.4 and Valbret, Moulines, and Tubach 1992; Peeters 1998; Peeters 2001).

The transformations applied in CATERPILLAR are only the adaptation of the mean loudness of the selected units to the target units, and the shortening of selected units that are longer than the target units. The shortening is only applied when the target times are specified explicitly. There is also a synthesis mode which uses the intrinsic durations of the database units. The reason for limiting the transformations to loudness change and shortening is that these two are the only ones that do not degrade the sound quality. Moreover, they do not necessitate a sound representation other than the sampled signal.

16.5.2 Concatenation

Concatenation smoothes the transition between adjacent selected units by applying the same transformations as listed above to a small part of the edges of the two units. When concatenating in a sustained part of a note or a phone, amplitude, pitch, and spectral jumps are reduced. When concatenating with an attack unit, this smoothing is not necessary.

For the scope of this work, a simple amplitude crossfade over 10 ms proved sufficient to provide a smooth concatenation.

For semi-automatic sound composition, it is a good idea to write the selection result in an editable format, for manual fine-tuning and correction. DIPHONE (section 18.6.2) is a program for just this task, and PSOLA transformation has just been added to it.

Chapter 17

Applications and Results

This chapter lists the applications that have been implemented using the CATERPILLAR framework, and other possible applications. From the most precise application of high level instrument synthesis (section 17.1), to the resynthesis of audio (section 17.2), to loop synthesis (section 17.3) and free synthesis (section 17.4), followed by the first steps of applying CATERPILLAR to speech synthesis (section 17.5). Other applications than in sound synthesis are possible, some are evoked in the conclusions in chapter 18.

17.1 High Level Instrument Synthesis

Because the CATERPILLAR system is aware of the context of the database as well as the target units, it can create natural sounding transitions. Information attributed to the source sounds can be exploited for unit selection, which allows high-level control of synthesis, where the fine details lacking in the target specification are filled in by the units in the database. As an example, a database was constructed from the *Sonata No. 1* and *Sonata No. 2* for solo violin (J.S. Bach's *Sonaten und Partiten*, over one hour of music, played by Yehudi Menuhin and Gideon Kremer. This corpus is described in detail in section 15.1.

The violin was chosen because it still presents difficulties for rule-based synthesis by signal models or physical models. See the work by Serafin and Smith (2000) and Smith (2003) for the state of the art in physical modeling of the violin and physical modeling in general. There are intricacies in the sound that are difficult to model, because of the many complex interactions between the strings, the bow, the fingers, and the body of the violin, above all in the note attacks and transitions. Moreover, even if the sound production could be perfectly modeled, the really hard problem is how to control the model to recreate all the fine details due to the expert playing by a professional violinist.

17.1.1 Sub-Segmentation

For the instrument synthesis application, we need finer grained units as the notes that result from score–performance alignment (see Part II). Sub-segmentation divides the basic note segment type, into smaller segments as shown in figure 17.1: The attack, sustain and release segments correspond to the classic subdivision of a note. The interest is that the attack phase bears most of the identity of the instrument, as shown by Schaeffer and Reibel (1967). The sustain phase is the stable part of a note, followed by the release phase where the note fades out. The sustain phase exists only for sustained instruments and playing styles. Any struck or plucked playing style makes the attack phase go right into the release phase.

The attack and release phases are currently set blindly to 100 ms at the start and before the end of the segment. More sophisticated methods exist to model the attack and release phases, such as the ones described by Jensen (1999).

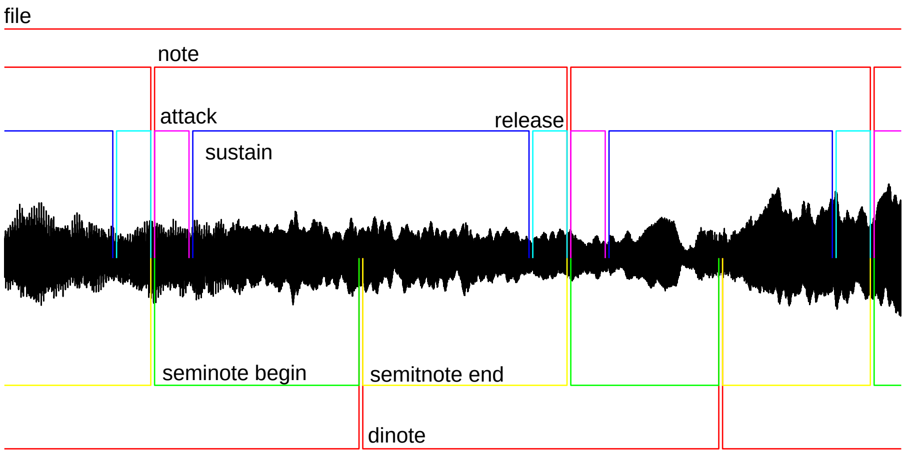


Figure 17.1: Sub-segmentation of a note into attack, sustain, release, seminotes and recombination as a dinote

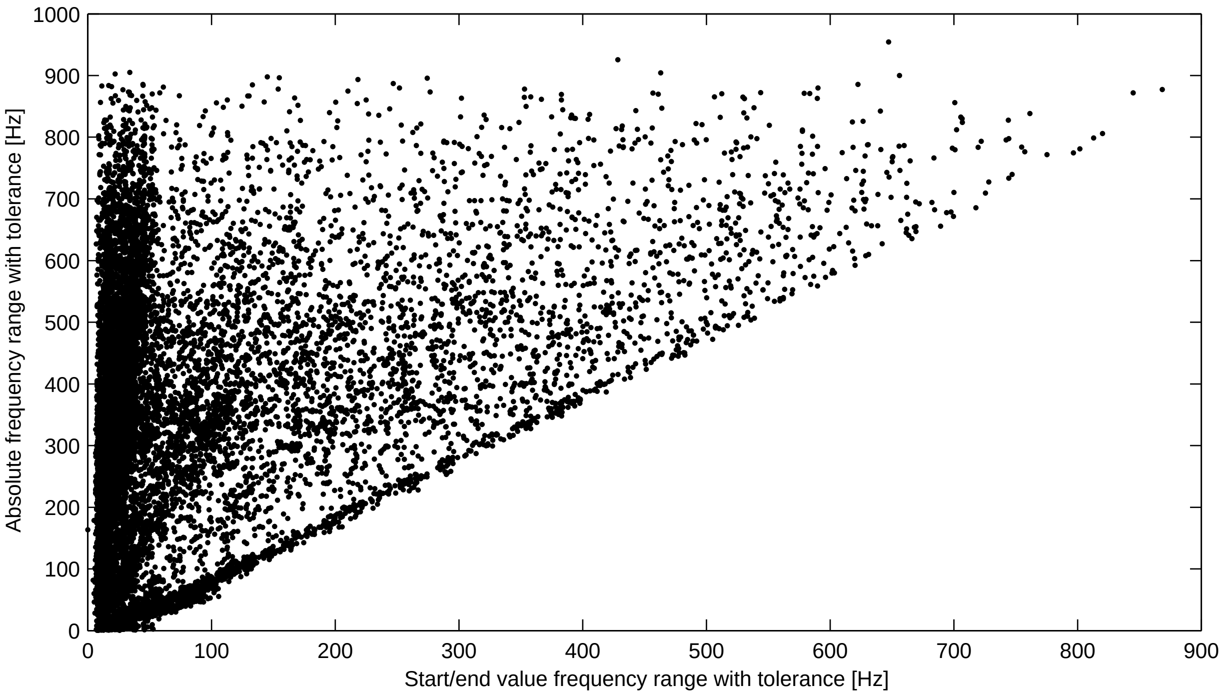


Figure 17.2: Absolute frequency range over start/end range for Corpus *Sonaten und Partiten*

The *dinote* segment type is analogous to a *diphone* from speech synthesis it is composed of two seminotes that are split in the middle of a note. This is usually within the sustain phase and so in a stable part of a note, where concatenation can take place with the least discontinuity.

17.1.2 Preselection of Appropriate Units

Due to the two sources of errors in the database explained in section 15.1 and shown in figure 15.4: errors in the segmentation and errors in pitch analysis, care has to be taken to select appropriate units for dinote instrument synthesis. Two sanity checks for dinote units verify that the pitch curve corresponds to what we expect: First, the pitch transition width (see section 11.2) must be smaller than 10 ms, to eliminate units containing more than two notes, and second, the absolute pitch range must not be greater than the range between the start and end value, plus one half tone of tolerance. The difference between the two ranges is used as a distance, to penalise units with outliers on the pitch curve, such as polyphonic units. The relation of start/end range versus absolute range is shown in figure 17.2.

17.1.3 Distance Functions and Weights

The target and selected features for dinote instrument synthesis are shown in figure 17.3. figure 17.4 shows the characteristic values of the selected units, and the target features as circles. The target distance functions, features, and their respective weights used for this example are given in table 17.1.

Descriptor	**Characteristic Value**	**Distance**	**Weight**	**Description**
unit type	mean	binary	10	To force a note-start seminote at the beginning and an end seminote at the end, with dinotes in between. A better choice would be to begin with a dinote starting from silence, but this would require a segmentation of the score into phrases, which is not yet done.
pitch (f0)	start/endval	Euclidean	100	The start and end values of pitch define our melody, assembled by the dinotes.
centroid	mean	Euclidean	0.001	This was to obtain an enrichment of the spectrum, but it has no apparent effect. If we augment the weight, the pitches are missed.
duration	mean	duration	0.01	We use this to favour units of at least 500 ms length.

Table 17.1: Features, target distance functions, weights for dinote synthesis

17.2 Resynthesis of Audio

A sound or phrase is taken as the audio score, from which a certain set of descriptors is used to constitute the synthesis target. We then resynthesise with these descriptors, e.g. the pitch, amplitude, and timbre characteristics, using units from the database.

Figure 17.5 shows an example of a selection according to an audio score (*Die Roboter* by *Kraftwerk*, sound example 8).

The target was segmented into short grain units, so that the selection had to consist also of short units. In order not to limit the available units, all units of a length larger than the grains were admitted for selection and then cut to the right size, but to obtain accurate descriptor values for the short snippet at the start, we used the start values of the descriptors.

The main feature with the highest weight was the *start value* of pitch (*f0 startval*). We can see that this feature is well followed by selection. The other features are weighted much less, and suffer large errors.

17.3 Loop Based Synthesis

In electronic dance music, a large part of the musical material comes from sampling CDs, containing rhythmic loops and short bass or melodic phrases. As the published CDs number in the thousands, each containing hundreds of samples, a large part of the work consists in listening to the CDs and select suitable material. A database would come handy, that stores loops by given

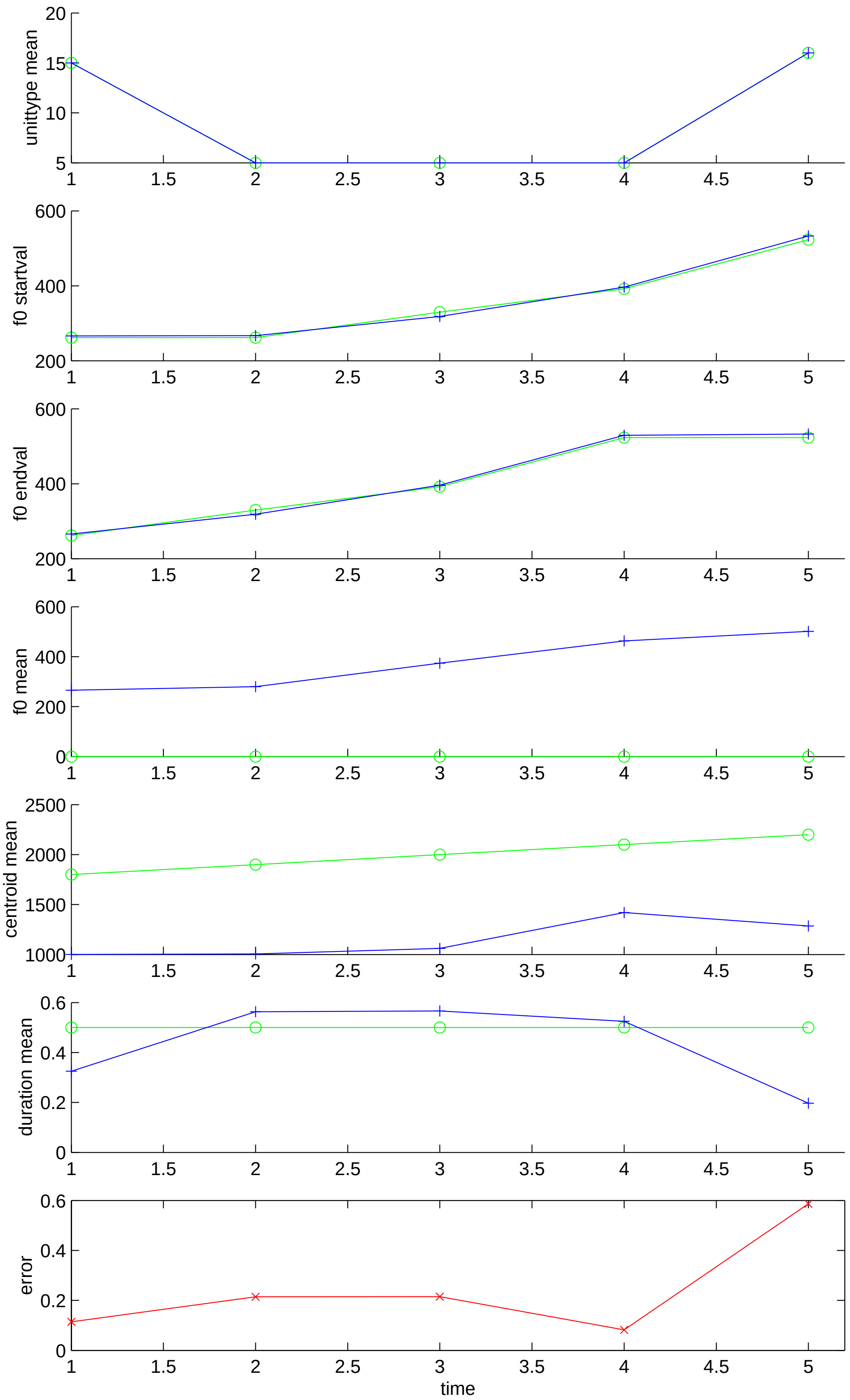


Figure 17.3: Selection of dinotes from violin corpus. The light circles are the points of the target in each of the descriptors, the dark crosses are the values of the selected units. The last sub-figure shows the error, i.e. the total unit distance.

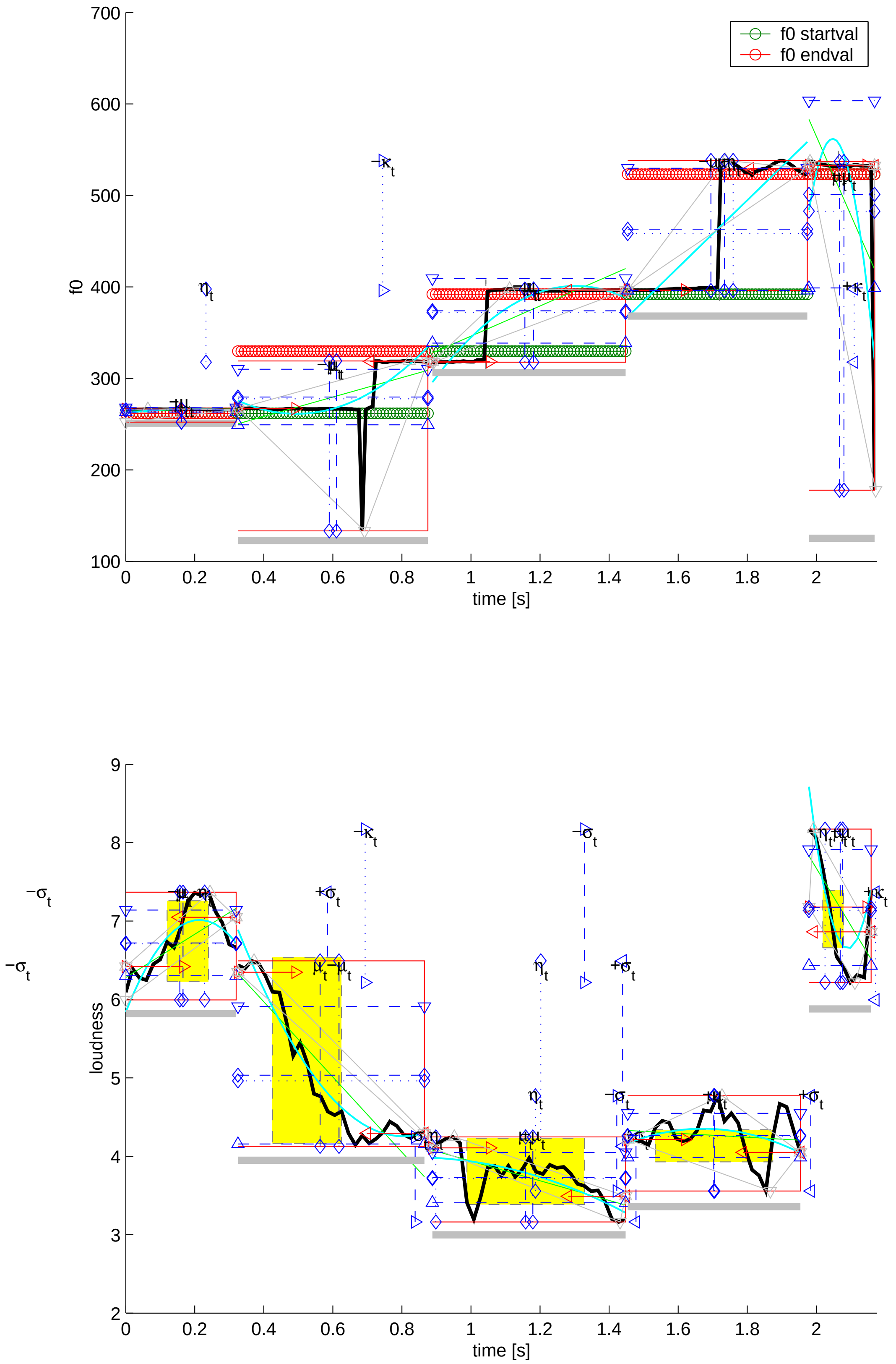


Figure 17.4: Selected units and characteristic values of pitch (f0) and loudness. The selection target for pitch (*f0 startval, f0 endval*) is drawn as circles. Loudness was not included in the target specification and shows therefore jumps for the two last units.

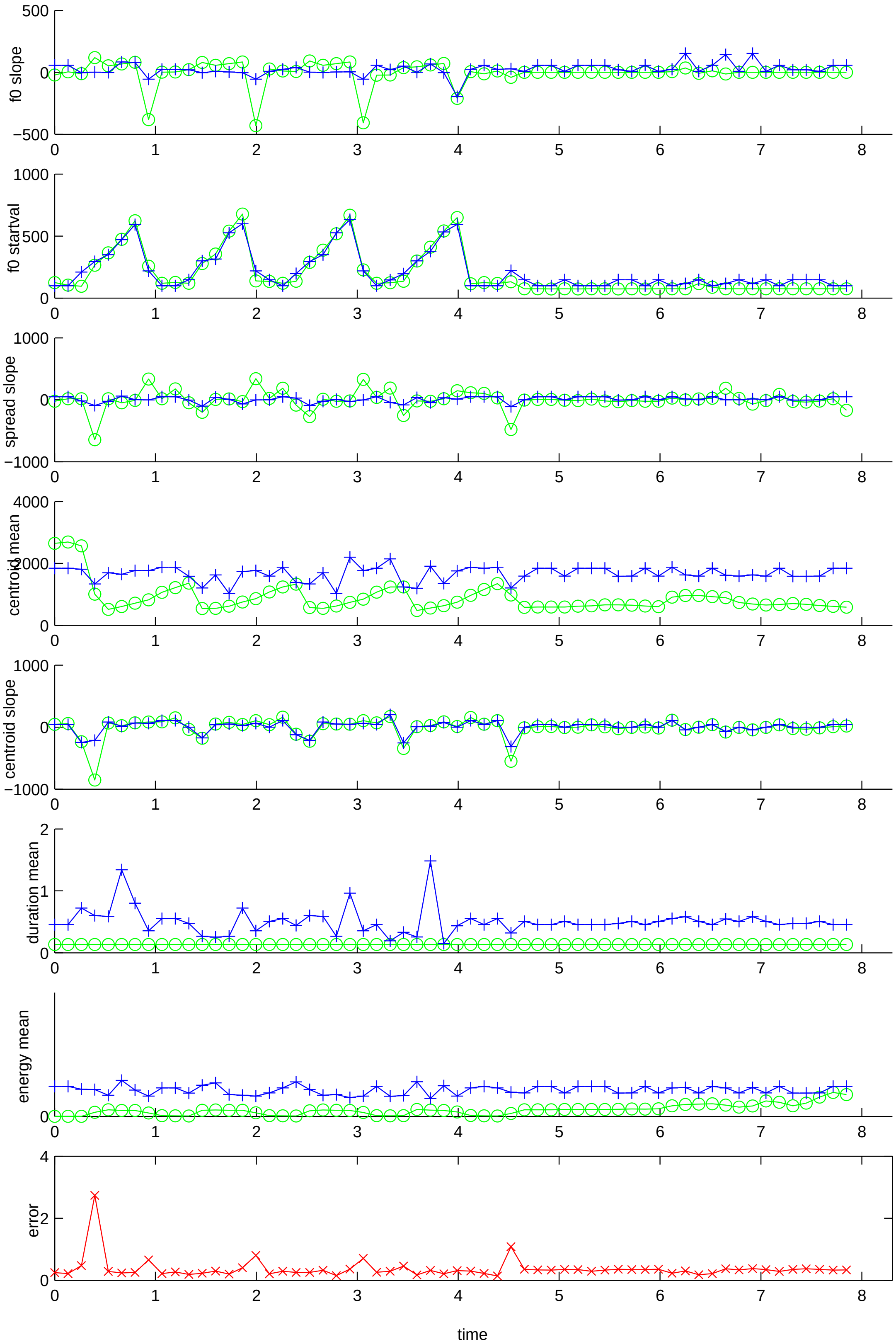


Figure 17.5: Selection from audio score by Kraftwerk. The light circles are the points of the target in each of the descriptors, the dark crosses are the values of the selected units. The last sub-figure shows the error, i.e. the total unit distance.

categories (rhythmic style and tempo (bpm) information is usually supplied by the CD description), and automatically analysed acoustic and perceptive descriptors, such as: high frequency content, sharpness, roughness, dissonance, etc., and certainly a descriptor for the percussiveness of a loop, evoked in section 18.2. The selection algorithm would then supply a sequence of loops satisfying a given dynamical evolution, the concatenation distance being controlled by the desire for continuity or change in the piece.

17.4 Free Synthesis

Free synthesis from heterogeneous sound databases offers a sound composer efficient control of the result by using perceptually meaningful descriptors to specify a target as a multi-dimensional curve. This type of synthesis is interactive and iterative. The Caterpillar system supports this by its graphical database browser (see figure 17.6) and the ability to freeze good parts from a synthesis and regenerate others, or to force specific units to appear in the synthesis. Our database contains various recordings of environmental, instrumental, voice, and electronic sounds, detailed in sections 15.1–15.3.

Free synthesis can be seen as a largely extended *granular synthesis* section 4.2.3, which offers the following advantages:

- We can direct the selection of grains according to the pre-analysed sound descriptors from the database, by composing a target trajectory through the whole database in the descriptor space, and not in the position of one sound file.
- In non-real-time mode, free synthesis takes care of the transitions between grains, allowing us to stipulate continuity in any of the descriptors, if desired.
- In real-time interactive browsing, we can visualise the content of the database and purposefully reach a certain position in the descriptor space, instead of searching through a soundfile by position. (This can be emulated by Caterpillar as well, since the position of each unit is a descriptor, too.)

17.5 Artistic Speech Synthesis

An interesting new project uses Caterpillar to recreate the voice of a defunct eminent personality to render a given limited text. For the 40$^{\text{th}}$anniversary of the death of Jean Cocteau, the film maker Chris Marker wanted to recreate the voice of the famous artist, writer, and film maker to read *La belle et la bête*, a text he had never recorded during his lifetime.

The goal here is different from fully automatic text-to-speech synthesis: highest speech quality is needed (concerning both sound and expressiveness), manual refinement is allowed. The text to be synthesised is fixed and of limited length, but of literary nature, so that the database has to match the power and expressivity of the text and has to be very large. The role of Caterpillar is to give the highest possible automatic support for human decisions and synthesis control, and to select a number of well matching units in a very large base according to emotional and expressive descriptors. The adaptations that had to be applied to the Caterpillar system to reach this aim were easy to integrate and are described in detail the following sections:

- Addition of linguistic descriptors for prosody (word stress, intra-word position), and phonetics (phoneme), which was easy, because the database is designed to allow extension of descriptors.
- Integration of new distance functions for the new descriptors into the unit selection algorithm
- Exploitation of the phoneme classes added to the database for faster unit search.

We chose the semiphone as basic unit because of the same reasons as for dinote synthesis (section 17.1.1): two semiphones can be recombined to a phone or to a diphone. The work by Blouin, Rosec, Bagshaw, et al. (2002) and Blouin (2003) confirmed this choice.

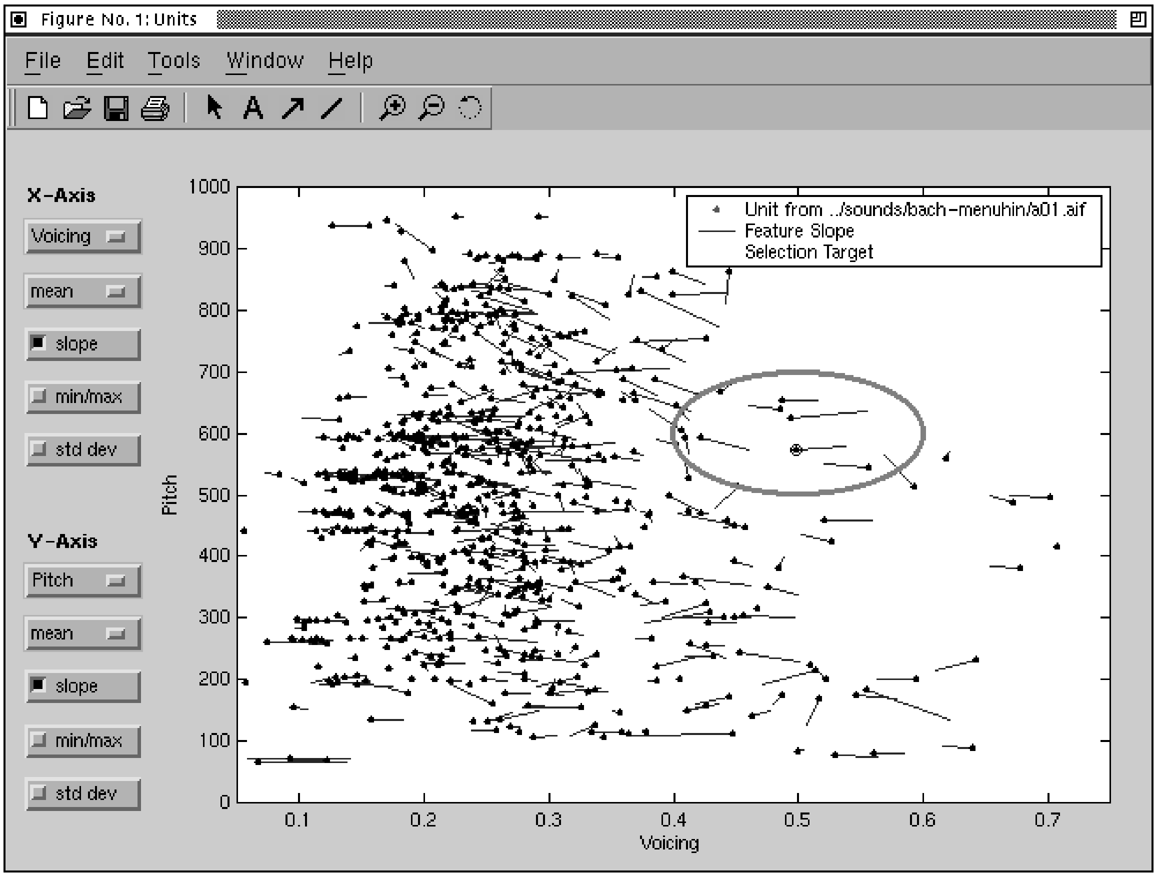


Figure 17.6: Database explorer feature view: Each point represents a unit, plotted according to two selectable characteristic values of two features. Various characteristic values can be displayed with the units, e.g. min/max, the standard deviation, or the mean slope (the short lines extending from the units). The ellipse serves to interactively select the units for real-time acoustic exploration of the database. The currently played unit within the ellipse is highlighted by a little circle.

17.5.1 Definition of Linguistic Descriptors and Categories

The linguistic descriptors added to CATERPILLAR express the phonological and syntactical context of the unit. They follow closely those proposed by Prudon (2003) and Blouin (2003). An overview is shown in figure 17.7 and details given in the following:

Duration
Unit duration (this is the standard CATERPILLAR descriptor).

Position
Unit position in its containing syllable, word, sentence.

Phoneme
Phoneme identity in X-Sampa.

Diphone
Diphoneme identity in X-Sampa.

Begin/end semiphone
True if unit is the semiphone at the begin or end of a phone.

Phonetic syllable/word
Phonetic syllable or word the unit comes from in X-Sampa.

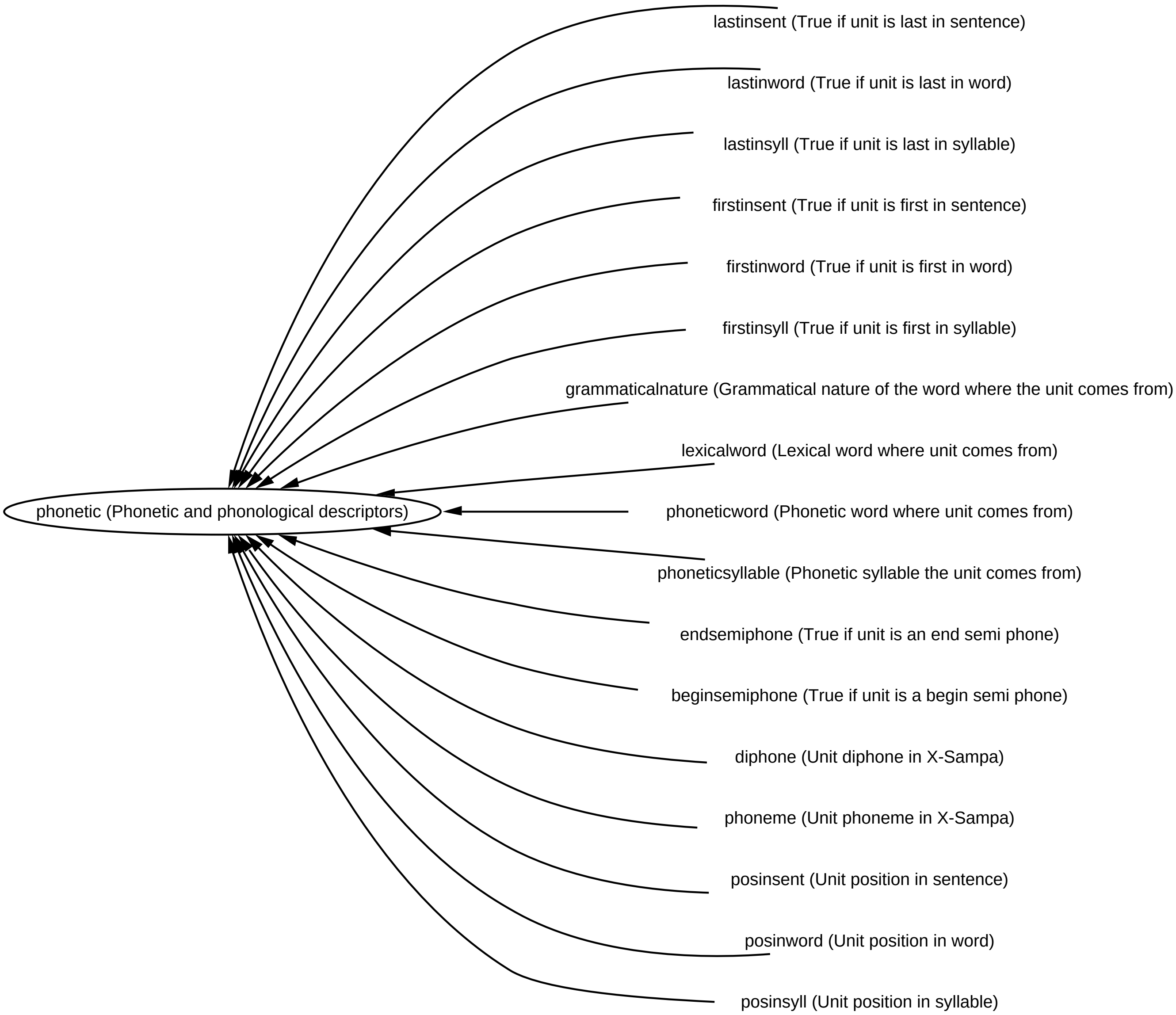


Figure 17.7: Phonetic descriptors for speech synthesis

Lexical word
Lexical word where unit comes from in ASCII.

Grammatical nature
Grammatical nature of the word the unit comes from.

First in syllable/word/sentence
True if unit is first in syllable, word, sentence.

Last in syllable/word/sentence
True if unit is last in syllable, word, sentence.

The The X-Sampa computer readable phonetic alphabet for French is explained in appendix F. For the phonetic class descriptors, the IPA phonetic classification (IPA 2003) was implemented as categories. They are detailed in appendix G. We can't stress enough how easy it was to integrate these descriptors and categories in Caterpillar, because the database is designed to allow easy extension of descriptors.

The new phonological distance functions necessary to exploit these high-level phonological descriptors are given by comparing the various positions, and by a distance matrix for the phoneme class categories.

17.5.2 Implementation

The necessary adaptations to Caterpillar for speech synthesis have been carried out with the help of Örsten Kärki during his internship described in (Örsten Kärki 2003).

As a bootstrap for database building, we needed precisely segmented units, and the linguistic information about the phonetic, phonological, and syntactical role of each unit. For the segmentation, a part of the source recordings were hand-segmented with the program Xspect. For the linguistic labeling and the text-to-phoneme conversion, the linguistic analysis stage of the multilingual text-to-speech system Euler[1] (Dutoit, Malfrère, Pagel, Mertens, Ruelle, and Gilman 1998; Bagein, Dutoit, Tounsi, Malfrère, Ruelle, and Wynsberghe 2001; Dutoit 2000) was used, configured to only save the linguistic information as a *Multi Layer Container* (MLC) without performing the synthesis.

An MLC is a frequently used data structure for text-to-speech synthesis (see chapter 3) to store the output of the various linguistic text analysis modules shown in figure 3.2. An example of an MLC for French is shown in figure 17.8.

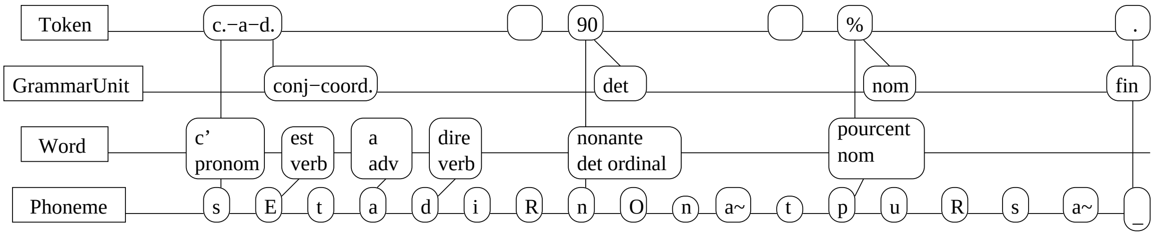


Figure 17.8: Example MLC for a French phrase

To preprocess and import the necessary source material into the database, the two segmental data streams from segmentation and MLC must be merged and added to the database unit by unit. To this end, two Perl scripts have been written. Their place in the import process is illustrated in figure 17.9. The merge script treats the conversion of the output formats of the external programs: For segmentation, the label data written by Xspect[2] (Rodet, François, and Levy 1996; Rodet and François 1996), Mbrolign[3] (Malfrere and Dutoit 1997b; Malfrere and Dutoit 1997a; Dutoit 1999), or Audacity[4] (Mazzoni and Dannenberg 2001) are read. For the higher level grammatical data, the MLC text output of Euler is parsed and flattened on a semiphone-by-semiphone basis. This means that, for each semiphone, all links between layers are traversed and the information found is copied to one SDIF frame per semiphone. These two streams are merged according to the phoneme label, and the flattened MLC is written in the SDIF type defined in appendix E.3, with the semiphone start times given by the segmentation data.

An unforeseen difficulty arose from the fact that the segmentation by hand also resulted in a new independent phonetic transcription of the source recordings. Due to ambiguous cases in the data and some remaining errors in Euler's text-to-phoneme conversion, the phoneme labels from segmentation differed slightly from the ones in the MLC. To avoid having to manually correct either the manual phoneme labels, or the MLC, the Unix *diff* utility was used, which generates the minimum number of editing commands (change, insertion, deletion) to change one string of symbols into the other one. In short, a symbolic text alignment was necessary in order to be able to merge the two data streams.

[1] http://www.tcts.fpms.ac.be/synthesis/euler
[2] http://www.ircam.fr/anasyn/DOCUMENTATIONS/xspect/index-e.html
[3] http://tcts.fpms.ac.be/synthesis/mbrolign/mbrolign.html
[4] http://www.audacity.org

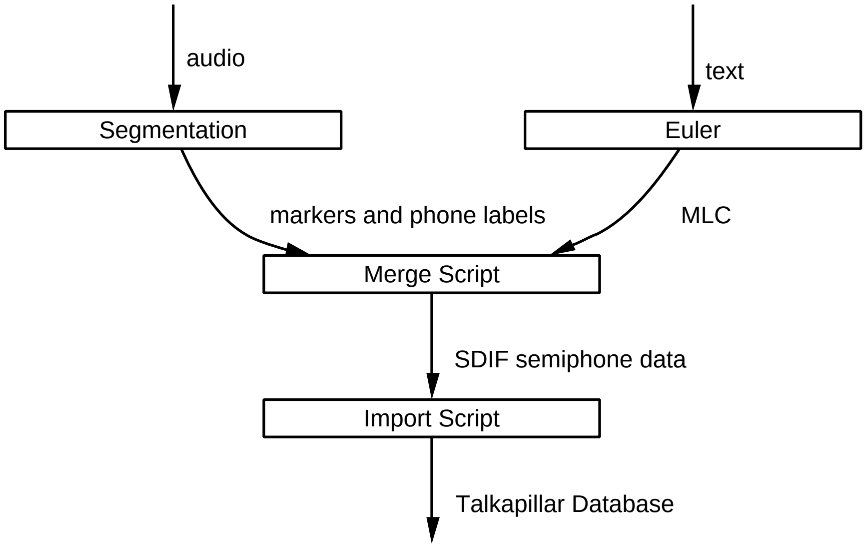


Figure 17.9: Preparation and importation of speech data into the database

Unfortunately, for financial and temporal reasons, the Cocteau project was abandoned, so that the database and the distance functions could not be refined.

Conclusion

"Would you tell me, please, which way I ought to go from here?"
"That depends a good deal on where you want to get to," said the Cat.
"I don't much care where—" said Alice.
"Then it doesn't matter which way you go," said the Cat.
"—so long as I get somewhere," Alice added as an explanation.
"Oh, you're sure to do that," said the Cat, "if you only walk long enough."

Lewis Carrol

Chapter 18

Conclusions and Future Directions

This chapter concludes this work and gives some directions for future research. It is organised according to the sub-topics that were treated: The alignment and segmentation (section 18.1), the descriptors and characteristic values (section 18.2), the database (section 18.3), selection (section 18.4, and synthesis (section 18.5), and the applications (section 18.6). Each section summarises the work that has been done, and shows paths for further development.

18.1 Alignment and Segmentation

Alignment (Part II) took a considerable amount of the research effort during this work, since although the methods are well known, and a vast literature exists, this refers usually to speech alignment, and some of the specific difficulties for music alignment had not been treated before. Even if so, the aim in the literature is usually content-based music information retrieval, where not a very high accuracy is required.

This work showed a continuous improvement from mean offsets of around 30 ms with simple methods that are also applicable to real-time alignment, to 24 ms with the more extensive analyses described in section 6.1, which comes close the human perceptual threshold of the detection of asynchronous events about 20 ms.

The recently started work of integrating a transient detector to refine the in-frame alignment promises to overcome the intrinsic in-frame imprecision by re-detecting the note onset within the aligned frame. Preliminary results showed an improvement of 3.5 ms for a corpus of synthesised test performances.

The work on reducing the memory requirements described in section 6.4 by the shortcut path and pruning was necessary to make alignment of longer pieces feasible at all, as is mandatory for the building of large unit databases for concatenative synthesis.

Work is under way to improve the accuracy and generality of score–performance alignment: The combination of score and beat alignment makes our DTW algorithm applicable to pop-music.

For sound sources where no score exists, we still need a better segmentation than the chopping up into arbitrary grains. For noises and effects sounds, we should investigate the use of a blind segmentation according to a "novelty" index of the signal, as described in (Hoskinson and Pai 2001; Cardle, Brooks, and Robinson 2003), which obtains a more natural segmentation than the grains.

18.2 Descriptors and Characteristic Values

The descriptors (chapter 10) are all the information we have at our disposal about the units, according to which we perform the selection. We use as a base the copious set of dynamic descriptors developed for music information retrieval, with some additions regarding categorisation of units.

For the raw descriptors to be usable for unit selection, we have to reduce them to the characteristic values for each unit (chapter 11), where we were able to find convenient models how to express the temporal evolution of a descriptor over a unit. Some of the characteristic values are specific to our applications, e.g. the start/end values and the transition width which informs us about the suitability of a unit for dinote selection.

Still, progress in the definition and automatic extraction of descriptors is highly desirable.

The recent progress in the establishment of a standard score representation format (see section 5.4) with *MusicXML* as the most promising candidate, means that we can soon overcome the limitations of Midi and make use of more high level information from the score, and thus perform unit selection on a higher level. This means exploiting musical context information from the score, such as dynamics (crescendo, diminuendo), and better describing the units (e.g. we'd know which units are trills, which ones bear an accent, etc).

We can already now derive more musical features from an analysis of the score, such as:

Harmony (chords, chord class, measure of consonance/dissonance, position in melodic phrase) This repletes further the concept of high-level synthesis by giving context information about the musical function of a unit in the piece, such that the selection can choose units that fulfill the same function.

Rhythm (position in the measure, relative weight or accent of the note) This applies to notes as well, but mainly to percussive sounds This information can partially be derived from the score but should be complemented by a beat tracking approach that analyses the signal for the properties of the percussion sounds. The prerequisite for this is that the automatic alignment incorporates percussion and is very precise (see section 18.1).

What is definitely needed is a descriptor for *percussiveness* of a unit. We can simulate that by looking for units with low AR-attack time and a decreasing energy, but there are more aspects that play a role in the perception of percussiveness. See Jaillet (2000) and Gouyon (2000) for some of these. Tzanetakis, Essl, and Cook (2002) answer this question for musical excerpts, by calibrating automatically extracted descriptors for the *beat strength* to perceptive measurements.

An interesting approach to the definition of new descriptors is the work by Zils and Pachet (2003) on automatic discovery of descriptors: A genetic algorithm evolves a formula using standard DSP and mathematical building blocks whose fitness is then rated using a cross validation database with data labeled by users. This method was successfully applied to the problem of finding an algorithm to calculate the *perceived intensity* of music.

18.2.1 Evaluation of Descriptor Salience

Thirty descriptors were developed and each one expressed in 31 *characteristic values*. This enormous number of parameters that could be used for selection carries of course incredible redundancies. However, as CATERPILLAR is to be used for musical applications, one can not know in advance, which features will be used. The aim is to give maximum flexibility to the composer using the system. Some applications described in chapter 17 only use a very small subset of these.

For the more precisely defined applications, a systematic evaluation of which descriptors and which characteristic values from the plethora described in chapters 10 and 11 are the most useful for synthesis, would be welcome, similar to the automatic choice of descriptors for instrument classification described in Peeters and Rodet (2003a) and Livshin, Peeters, and Rodet (2003).

18.3 Improvements of the Database

The database is the core of any data-driven or data-based system. Its flexibility decides if further development of the system is easily possible or blocked by a rigid structure. Its reliability and performance is crucial for the daily work with the system. In our system, the step was made to use the proven technology of relational databases, developed for large-scale data-based applications, instead of developing a specific database from scratch. The flexibility of the relational framework posed interesting problems of modeling the section of reality that we wish to map onto the database. Innovative answers have been found and described in chapter 12 that are general enough to be applicable to other applications dealing with sound, classes, segments, and descriptors.

The development of the CATERPILLAR database took some time, and there were some surprises of very slow query execution, when the SQL optimiser managed to find the most complicated way to execute a query. The remedy is the reformulation of the query or the introduction of appropriate indices in the database. This is due to the declarative nature of SQL, where *what* you want is specified and not *how* to obtain it, because the latter is figured out by the database.

However, the advantages of using a relational database prevail: SQL enables us to clearly define the data model that is embodied in the database schema, makeing explicit all relationships between the entities. The daily work showed that using an SQL database for CATERPILLAR was a quantum leap forward for the ease and security of data handling, with the automatic consistency checks according to the relations defined in the data model.

So the question about the use of a relational database "Would you do it again?" is clearly to be answered with "Yes, even more!" More of CATERPILLAR's functionality should be handled in the database. Ideally, the whole database-interface should be implemented in the procedural PL/pgSQL language, to be independent of the MATLAB environment. Unfortunately, this extension language became available only after large parts of the database interface had already been developed. Finally, even the unit selection algorithm could be implemented within the DBMS, maybe even running on the server, to obtain the best performance. Then, levels of performance could be obtained that are at least as good as with a database specifically developed for an application, or even higher levels, since the DBMS provides various optimisation strategies concerning disk block caching, etc.

Specific plans for improving the handling and performance of the database schema have already been detailed in section 14.4 and will be implemented shortly to promote the use of CATERPILLAR for other applications. In general, the user interface to the database is still very rudimentary and needs to become more friendly.

18.4 Unit Selection

When the database is the heart of a data-driven synthesis system, the unit selection algorithm is its brain. Continuing in the analogy, the distance functions are the senses of our system. In chapter 16, the simple but usable Euclidean distance functions are presented, that serve also for concatenation distances if continuity is required. They are extended to incorporate explicit context, which is useful for high-level synthesis. Preselection is an important step to reduce the amount of data to be treated. We saw then the proven method of unit selection as a best-path search through a network of units, with the possible optimisation of pruning the path to minimise concatenation cost calculation. The reformulation of unit selection as a constraint satisfaction problem (CSP) described in section 16.4 allows to combine the requirements for creative synthesis with the proven unit selection principle from speech synthesis, keeping maximum flexibility.

One important open research question is how to map the descriptors we can automatically extract from the sound data to a perceptive similarity space that allows us to obtain distances between units.

18.4.1 Data-Driven Optimisation of Unit Selection

One possibility that has not been explored in this work is to exploit the data in the database to analyse the natural behaviour of an underlying instrument, which enables us to better predict what is natural in synthesis.

18.4.1.1 Learning Distances from the Data

Knowledge about similarity or distance between high-level symbolic unit descriptors can be obtained from the database by an acoustic distance function, and classification. For speech, with the regular and homogeneous phone units, this is relatively clear, but for music, the acoustic distance is the first problem: How do we compare different pitches, how units of completely different origins and durations? Moreover, for speech, Blouin (2003) reports worse results for a distance function trained from the data, than with a training using a priori linguistic knowledge, concluding that the data-driven distance measure needs further research.

18.4.1.2 Learning Concatenation from the Data

Another interesting application of a corpus of instrument sound units is to learn the concatenation distance function from the data. This is performed by statistical analysis of the descriptors of pairs of consecutive units in the database. The set of each unit's descriptors defines a point in a high-dimensional descriptor space D. The natural concatenation with the consecutive unit defines a vector to that unit's point in D. The question is now if we can generalise this vector field in a way that, given any pair of points in D, we can obtain a measure to what degree the two associated units concatenate like if they were consecutive.

The problem of modeling a high-dimensional vector field becomes easier if we restrict the field to clusters of units in a corpus and calculate the distances between all pairs of cluster centres. This will provide us with concatenation distance matrix between clusters that can be used as a fast lookup table for unit selection. This discretisation allows us to use the database for synthesis: we model the probabilities to go from one cluster of units to the next, depending on which unit we came from, generating a sequence of units that will respect the typical articulations taking place in the database source.

18.4.1.3 Learning Weights from the Data

Finally, there is a large corpus of literature about automatically obtaining the weights for the distance functions by search in the weight-space with resynthesis of natural recordings (Hunt and Black 1996; Macon, Cronk, and Wouters 1998). A performance optimised method, applied to concatenative singing voice synthesis, is described in (Meron 1999).

All these data-driven methods depend on an acoustic or perceptual distance measure that can tell us when two sounds "sound the same". Again, for speech this might be relatively clear, but for music, this is itself a subject of research in musical perception and cognition.

18.5 Synthesis

Our very simple crossfade synthesis is enough for the first steps (or crawls) of concatenative sound synthesis that CATERPILLAR makes. Eventually, one would have to apply the research from speech synthesis about reducing discontinuities evoked in section 3.3.2, or use a more advance signal model.

An open question is if hybrid concatenation of segments using different signal models like additive sinusoidal plus noise and PSOLA (Peeters 2001), or spectral and wave representation (Levine 1998) is possible. I.e. can we find an algorithm that combines the four different parts of the two units to concatenate, even though the separation between the sinusoidal and the PSOLA parts might not be at the same frequency?

CATERPILLAR, as a research system, works with mono sounds only. Extension to stereophonic or multi-channel sounds requires some program adaptation, but there are also some new descriptors about the spatial features of the sound to be defined. The more obvious ones would be stereo position or balance, and stereo width.

To make unit selection synthesis possible in real-time (for a real browsing of the database), we can think of a system that preloads the first sample block of each unit that falls in the preselection, to be able to play it immediately while scheduling the disk access to read the remaining signal. For a corpus of 80000 units, this makes roughly 40 MB with 256 samples preloaded per unit, giving us 5.8 ms to load the rest of the data.

18.6 Applications

The work on the applications of data-driven sound synthesis has been more in breadth than in depth: none of the applications has been pursued and researched until the end, rather, the groundwork was laid with the CATERPILLAR system, and several applications were implemented in a prototypical manner, as proof of concept. This is why the acoustic result of the syntheses sounds sometimes quite rough. To go in depth, for the instrument synthesis, one would have to conduct listing tests, comparing different parameterisations of unit selection concatenative synthesis, and compare the listener preferences with other synthesis techniques (sampling, physical modeling, actual recordings). For free synthesis and loop synthesis, an evaluation of the requirements of composers and musicians, and usability studies should be performed.

For artistic speech synthesis described in section 17.5, the CATERPILLAR database proved its flexibility and generality of design: specific speech structures could be easily incorporated. Usually, speech synthesis systems use a special data structure called *multi-layer container* (MLC). The descriptors and unit tree represent already a large part of the information contained in an MLC, the missing information (layers) can be easily added. Thus, we can argue that our database can be seen as a superset of an MLC with the added advantages of relational databases.

18.6.1 Usability of Selection

We have see the advantages of constraint-based unit selection in section 16.4. We gained much flexibility in formulating musical criteria for selection. However, this formulation was still embodied in MATLAB code. For future use by composers and musicians, the constraints governing the unit selection algorithm should be formulated in a domain-specific language. This approach has been pursued in the constraint solver package for OPENMUSIC (Truchet, Agon, Assayag, and Codognet 2001; Truchet, Agon, and Codognet 2001; Truchet, Assayag, and Codognet 2001).

18.6.2 Links with the DIPHONE Program

DIPHONE (Rodet and Lefèvre 1997) is a graphical sound composition environment which controls additive synthesis, CHANT, and PSOLA. The central concept of the program is that of assembling diphones: For the program, a diphone is a segment of a parametric description of sound. When diphones are combined to sequences, the parameters in the overlapping parts between them will be interpolated, allowing, for instance, morphing between completely different sounds.

There are two possible directions how to combine CATERPILLAR and DIPHONE:

1. Concatenative synthesis could become another synthesis model controlled by DIPHONE, whose graphical editing facilities would be used to compose a target specification for CATERPILLAR.

2. More interestingly, CATERPILLAR could output a sequence of units which are the result of unit selection in the format of a DIPHONE sequence, so that the sequence could be edited and refined in DIPHONE. This touches mostly on the transformation which is possible in DIPHONE, using an additive representation or PSOLA.

18.6.3 Adaptive Target Re-segmentation

Due to its iterative nature, CSP unit selection section 16.4 makes possible one more important improvement impossible with path search unit selection: the target could be automatically re-segmented when the unit selection cost C^s remains too high. When, after a certain number of iterations of the CSP unit selection algorithm, a unit's cost cannot be improved, and the cost is above a resegmentation threshold Θ_r, we try to split the target unit into two new units, in the hope that we can find two smaller units that together form a better match for the original target unit.

This would be especially well applicable to resynthesis of audio, since then segmentation errors of the target sound have less influence on the result because they can be corrected. We could even imagine to start with only one target unit for the whole target sound and have the unit selection algorithm develop the match and the segmentation. Note that a sort of backtracking, i.e. undoing of a target unit split, is automatically included by favouring the selection of contiguous units.

The remaining questions are how to determine the split point (by binary separation, or by trying multiple position and keeping the best one), and if it would be promising to also try to shift the target unit boundaries without introducing new ones.

18.6.4 Evaluation

Listening tests have not been done, because the synthesis result is still too rough. It is not clear which question to ask the subjects in such a test, contrary to speech synthesis where intelligibility and naturalness are rated. For the instrument synthesis applications, one could ask for preference of this over other synthesis methods, e.g. sample based or physical modeling. (But what with the differences in control of the synthesis, which might have a greater influence on the results than the sound.) For the other applications, which are geared more towards a composer, the only quality criterion is how useful CATERPILLAR is in the creative process. It is hardly possible to set up a controlled test for this.

18.6.5 Going Further

Concatenative text-to-speech synthesis systems can use different voices by switching the database. This changes of course the timbre of the voice, and also the general mode of speech (neutral, informative, etc.), and certainly the identity of the speaker shine through in the synthesis. Analogously, for concatenative instrument synthesis, using databases from one recording session of one musician, we reproduce the sound of the particular instrument and the acoustic environment, but what happens with the style of the particular musician? Can we recognise Gideon Kremer or Yehudi Menuhin other than from the sound of their instruments or the recording (heavily biased by the characteristics of the different epochs the recordings were made—see section 15.1).

It is an interesting question to see what remains of the personality or articulation of the musician. For the moment, the synthesis is too uneven to be able to convey a style, i.e. the audible artefacts spoil the perception of the musical intention. However, even without synthesis the two parallel corpora in the database can be exploited for performance study.

18.7 General Conclusion

The aim for the development of the CATERPILLAR system was to reach a high genericity in the structures and methods used, but to prove with concrete applications that these can be specialised to solve musical and sound composition problems. In order to achieve this aim, the state of the art in many fields had to be combined, and sometimes developed further, to set up a system to test the hypothesis that concatenative synthesis based on unit selection is a viable synthesis model, and to explore what can be done with a large amount of data for sound synthesis.

Finally, the reader must have remarked that no actual definition of the term *data-driven* was given in this work, but the many examples described here hopefully showed the meaning: Data-driven methods are trainable, use data-based analyses, and are stochastic, in short: they use the information in the data, not rules.

Unfortunately, applying these criteria to our own work, we must admit that CATERPILLAR only qualifies for data-based synthesis, and the title's promise of data-driven concatenative sound synthesis has not been completely fulfilled in this work. However, now that the base of the system is laid down, many further applications become possible.

Figure 15.4 shows the dilemma of data-driven synthesis: In order to have enough coverage of units and contexts, we need large databases, but the necessarily automatic methods to create them introduce noise and errors in the data.

Concatenative text-to-speech synthesis has, after 15 years of research, now become a technology mature to the extent that all recent commercial speech synthesis systems are concatenative based on unit selection (see section 3.2). This success is also due to the database size of up to 10 hours of speech, a size we did not yet reach for musical synthesis.

Moreover, this database size is needed to adequately synthesise just one "instrument" — the human voice. What we set out for with data-driven concatenative sound synthesis is synthesising a multitude of instruments.

For musical synthesis, we stand at the same position speech synthesis stood 10 years ago, with yet too small databases, and many open research questions.

Appendix

"Is that all?" said Alice.
"No," said the Caterpillar.

Lewis Carrol

Appendix A

The CATERPILLAR Database Schema

The actual tables and attributes of the CATERPILLAR database schema are given here in a syntax close to SQL. They are ordered by the commonly used distinction between base tables (A.1) and working tables (A.2), an additional sub-group of the latter of data tables (A.4), and SQL's lack of forward references.

This documentation is automatically extracted from the SQL source files by the script *docsql*, documented in appendix D.2. This script also generates the UML class diagrams in figures 14.4 and 14.5.

A.1 Base Tables

The tables Descriptor (A.1.1), IsA (A.1.3), Symbol (A.1.2) and FeatureAnalysis (A.1.4) represent the basic structural data of the CATERPILLAR database. They change the least often.

A.1.1 Table Descriptor

Table Descriptor represents one sound descriptor (see chapter 10), which is the "datatype" of the actual unit data values in table UnitFeature (A.4.1).

Descriptor

ftid	**integer**	*unique index*[1]
name	**varchar**	*name of descriptor*
type	**char**	*type of descriptor: feature, category, corpus or group (of features)*
isroot	**bool**	*Is this class the root of a hierarchy, i.e. a categorisation aspect such as "source" or "mode of production"? This flag applies mainly for classes, but feature descriptors use the hierarchy, too, for logical grouping. N.B.: There can be more than one root in a hierarchy.*
unit	**varchar**	*unit of measurement: NULL for groups, bool for categories and corpora, because, seen as a feature, they represent class membership.*[2]
datatype	**char**	*data type, can be real, integer, boolean, or symbol (the value in table UnitFeature (A.4.1) is then the syid key of table Symbol (A.1.2))*
dynamic	**bool**	*dynamic descriptors (analysed ones like pitch, energy, etc.) change over the course of the unit and therefore have characteristic values (see CharacteristicValues (A.4.2)), static descriptors (category membership, integer, score and symbol values like midi-pitch) only have an entry in UnitFeature (A.4.1).*
description	**text**	*description*

[1] The unique index attribute is called ftid as in "feature type" for historical reasons.

[2] It is posible to model a fuzzy category membership by setting the datatype to 'real' and adding a value between zero and one to table IsIn (A.2.4). However, this is not necessary for our application.

The following views on table Descriptor (A.1.1) simulate the class specialisation hierarchy, because true inheritance is not available consistently in SQL.

A.1.1.1 View FeatureType

The view FeatureType is the subclass of table Descriptor (A.1.1) containing only numerical features, i.e. no category or corpus memberships.

A.1.1.2 View Corpus

The view Corpus contains all entries of table Descriptor (A.1.1) that can form a corpus, i.e. categories or true corpora.

A.1.1.3 View CorpusOnly

The view CorpusOnly is the subclass of table Descriptor (A.1.1) containing only true corpora.

A.1.1.4 View Category

The view Category is the subclass of table Descriptor (A.1.1) containing only categories.

A.1.2 Table Symbol

Lookup table for symbol values (analogous to an *enumeration* in C): For an ftid, we store pairs of syid, text.

Symbol

ftid	**integer**	*id of associated feature type, key of table Descriptor (A.1.1)*
syid	**integer**	*symbol number*
name	**text**	*symbol text*

A.1.3 Table IsA

The table IsA defines the inheritance hierarchy of categories or corpora (represented by entries in table Descriptor (A.1.1)). For convenience, also the logical grouping of table FeatureType (A.1.1.1) descriptors is expressed by links with type "group".

A pair of (ftidchild, ftidparent) means that each unit in the child category belongs also to the parent category.

IsA

ftidchild	**integer**	*id of child category or corpus*
ftidparent	**integer**	*id of parent category or corpus*
type	**char**	*flag whether the parent relationship was given by the user, or calculated automatically by transitive closure, or serves simply to group related descriptors*[3]
distance	**float4**	*node distance in category tree: direct (given) inheritance has distance 1, other nodes have the mininum of the sum of distances*

A.1.3.1 View IsaView

Convenience view to inspect the isa relationship with category names.

[3]This is needed to display only the important edges in the inheritance tree.

A.1.4 Table FeatureAnalysis

Table FeatureAnalysis describes an analysis program, and its parameters, called to compute unit data.

FeatureAnalysis

faid	**integer**	*unique index*
componentname	**text**	*description of the analysis component = program*
compversion	**text**	*program version*
comparguments	**text**	*string of arguments*
comment	**text**	*comment*
dateadded	**timestamp**	*the date this feature analysis was added, to define the notion of 'latest feature analysis'*

A.1.5 Table Analyses

Table Analyses records which feature types a feature analysis outputs, (possibly many) and how to access this descriptor's data in the output SDIF file.

Analyses

faid	**integer**	*The FeatureAnalysis (A.1.4) that*
ftid	**integer**	*analyses this Descriptor (A.1.1).*
sdifselection	**text**	*If output is in SDIF format, the SDIF selection specification (see section 13.4.2) tells us how to access this descriptor's data in the output file.*

A.1.5.1 View AnalysesView

Convenience view of table Analyses (A.1.5) which lists the names of the feature analysis program and the descriptor.

A.2 Working Tables

The working tables BaseFile (A.2.1), Unit (A.2.3), IsIn (A.2.4), ParentUnit (A.2.5) and NextUnit (A.2.6) are changed in the daily work with the Caterpillar database when adding soundfiles and their units boundaries.

A.2.1 Table BaseFile

The table BaseFile contains a reference on a sound file and its attributes, or to a *feature file* with analysis data. It can also describe a virtual file that only serves as a container for a sequence of target units.

BaseFile

bfid	**integer**	*unique index*
path	**text**	*access path to file name (can be NULL for virtual files)*
name	**text**	*basename of path without extension, used for name searches by the user*
type	**char**	*type of file (sound, feature, or virtual)*
format	**varchar**	*file format*
comment	**text**	*file description*
date	**timestamp**	*creation date of file*
Encoding information for sound files or data files		
datarate	**float8**	*sampling rate or frame rate*

numchannels	**integer**	
duration	**float8**	
numsamples	**integer**	*number of total samples*
databits	**integer**	*bits per sample or per data element*
datatype	**char**	*representation of sample or data (int, float, etc.)*

A.2.1.1 View SoundFile

The view SoundFile is the subclass of table BaseFile (A.2.1) containing only sound files.

A.2.1.2 View FeatureFile

The view FeatureFile is the subclass of table BaseFile (A.2.1) containing only feature files.

A.2.1.3 View VirtualFile

The view VirtualFile is the subclass of table BaseFile (A.2.1) containing only virtual files, i.e. files that serve to group units, e.g. for synthesis targets.

A.2.2 Table AnalysisRun

The table AnalysisRun records that a base file was analysed by a feature analysis program producing a feature file. Its primary key aid is used in tables Unit (A.2.3), UnitFeature (A.4.1) and CharacteristicValues (A.4.2) to distinguish between different segmentations and the same descriptor data types produced by different analysis programs.

AnalysisRun

aid	**integer**	*key of the analysis run*
bfid	**integer**	*key of the analysed sound BaseFile (A.2.1)*
ffid	**integer**	*key of the generated feature BaseFile (A.2.1), can be NULL if data is calculated from other sources*
faid	**integer**	*key of the used FeatureAnalysis (A.1.4) (ON DELETE is not set to CASCADE, because we don't want to loose all the work just because we threw out some analysis program entries)*
date	**timestamp**	*timestamp of the analysis*

A.2.2.1 View AnalysisRunView

Convenience view of table AnalysisRun (A.2.2) with expanded file and analysis names.

A.2.3 Table Unit

The data defining the units to be selected and concatenated are stored in table Unit. The actual feature values for each segment are stored in tables UnitFeature (A.4.1) and CharacteristicValues (A.4.2). The feature value would normally be the mean value of the feature in an analysis file referenced in table BaseFile (A.2.1), averaged over the unit.

Unit

uid	**integer**	*unique index*
bfid	**integer**	*id of the BaseFile (A.2.1) the segment is from*
starttime	**float8**	*start time in the sound file in seconds*
endtime	**float8**	*end time in the sound file in seconds*
type	**integer**	*type of unit (syid for Symbol with ftid 1)*
aid	**integer**	*the AnalysisRun (A.2.2) which produced the segmentation (ON DELETE CASCADE allows to throw out a batch of units)*

A.2.3.1 View UnitView

Convenience view of table Unit (A.2.3), listing units with basefile names.

A.2.4 Table IsIn

Table IsIn records the membership of a unit to a category or corpus. We don't need to keep the transitive closure here, because this is already done in the IsA (A.1.3) hierarchy amongst categories, i.e. we can find out with one query all the categories a unit belongs to. This is also saving space since there are many more units than categories.

IsIn

uid	**integer**	*key of adhereing unit in table Unit (A.2.3) (INITIALLY DEFERRED is necessary for automatic generation of representant units by insert triggers: the unit is created in the same transaction, so an immediate constraint would complain)*
ftid	**integer**	*key of category or corpus (represented by entries in table Descriptor (A.1.1))*

A.2.5 Table ParentUnit

The table ParentUnit defines the containment hierarchy between units from table Unit (A.2.3). A child unit inherits the category or corpus membership from its parent. On creation of a base file, one root unit is created that represents the whole basefile, such that the basefile categories are automatically propagated down to the children. Note that this relationship is in general not a tree, but a graph, since overlapping units that share sub-units are allowed.

ParentUnit

uidchild	**integer**	*key of child unit*
uidparent	**integer**	*key of parent unit*

A.2.6 Table NextUnit

The table NextUnit records the temporal order of units from table Unit (A.2.3) in a basefile. This speeds up the detection of a natural concatenation in the calculation of the concatenation distance. Note that one unit can have multiple next or previous units, e.g. one note unit has the following note unit and its attack sub-unit as next units.

NextUnit

uidprev	**integer**	*key of preceding unit*
uidnext	**integer**	*key of following unit*

A.3 Corpus Related Tables and Views

These tables are generated by the CATERPILLAR database from the category memberships and unit data. They are used to speed up data access for selection.

A.3.1 Table UnitInCorpus

Table UnitInCorpus is the materialisation of the view CorpusUnits (A.3.3), which unfolds the linked IsA (A.1.3) and IsIn (A.2.4) hierarchies.

UnitInCorpus

cid	**integer**	*key of the corpus, represented as a Descriptor (A.1.1)*
uid	**integer**	*key of the unit*

A.3.2 View DirectCorpusUnits

Find units directly in corpus and their child units.

A.3.3 View CorpusUnits

The view CorpusUnits(uid, ftid) unfolds the two linked CATERPILLAR hierarchies for units and categories: It contains in its rows all units uid in the corpus ftid and all child units. Each unit appears with all non-root categories it belongs to accoring to the IsIn (A.2.4) relationship (because we keep the transitive closure).

Thus, to get all units in a category, filter with its id:

```
SELECT uid FROM CorpusUnits WHERE ftid = mycorpus;
```

A.3.4 View DirectCorpusFiles

Get all tuples for basefiles and corpora they directly belong to

A.3.5 View CorpusFiles

Get all tuples for basefiles and corpora they belong to hierarchically

A.3.6 View CorpusSummary

Create summary table listing number of files, number of units and total length of files for each corpus.

A.4 Data Tables

The working tables UnitFeature (A.4.1) and CharacteristicValues (A.4.2) contain the bulk of the data in the CATERPILLAR database: The analysed features of the units and their characteristic values. The views UnitData (A.4.3), CorpusUnitData (A.4.4) and BasefileUnitData (A.4.5) perform the join of the two data tables, optimised for selecting all units from a corpus, or a basefile. This is necessary, since the LEFT JOIN between UnitFeature and CharacteristicValues is very expensive, so the selection has to be done as deep down in the view as possible, to eliminate the unused units before they are joined.

A.4.1 Table UnitFeature

The table UnitFeature stores the mean value of a feature for a unit. It is complemented by table CharacteristicValues (A.4.2) for non-constant features. We need to store both the feature analysis and the feature type, since one feature analysis can calculate several features, which can be in one single feature file.

UnitFeature		
uid	**integer**	*key of unit from table Unit (A.2.3)*
ftid	**integer**	*key of feature type from table Descriptor (A.1.1)*
aid	**integer**	*key of the analysis run AnalysisRun (A.2.2) that calculated the feature (which stores the FeatureAnalysis (A.1.4), sound and data BaseFile (A.2.1)) (ON DELETE CASCADE allows to throw out a batch of unit data, INITIALLY DEFERRED is necessary because we add the analysisrun and the unit data in the same transaction, so the data is visible only at its end)*
Only one of the following is used, depending on the descriptor data type given in Descriptor (A.1.1):		
intval	**integer**	*integer value, bool value or symbol index*
realval	**float4**	*float value of descriptor or arithmetic mean of feature for continuous descriptors*
textval	**text**	*string value (or copied symbol)*
arrayval	**float4**	*array value*

A.4.2 Table CharacteristicValues

This table complements the table UnitFeature (A.4.1) with the characteristic values other than mean, discussed in chapter 11. It is only used to describe the continuous descriptors of each unit. The keys are the same as in UnitFeature.

CharacteristicValues		
uid	**integer**	*key of unit from table Unit (A.2.3)*
ftid	**integer**	*key of feature type from table Descriptor (A.1.1)*
aid	**integer**	*key of the analysis run AnalysisRun (A.2.2) that calculated the feature*
Range of the feature value (see section 11.1)		
geomean	**float4**	*geometric mean of feature value over the unit (the arithmetic mean is the realval of table UnitFeature (A.4.1))*
std	**float4**	*standard deviation*
startval	**float4**	*value at start of unit*
endval	**float4**	*value at end of unit*
minval	**float4**	*minimum value of feature*
maxval	**float4**	*maximum value of feature*
absrange	**float4**	*absoulte range of feature*
slope	**float4**	*slope of linear approximation*
curve	**float4**	*curve of 2^{nd}order polynomial approximation (see section 11.4)*
residual	**float4**	*normalised residual of 2nd order polynomial fit*
Temporal distribution of feature over unit (see section 11.2)		
t_mean	**float4**	*temporal center of gravity*
t_antimean	**float4**	*temporal center of antigravity*
t_std	**float4**	*temporal standard deviation*
t_skewness	**float4**	*temporal skewness*
t_kurtosis	**float4**	*temporal kurtosis*
AR-envelope: attack time until maximum value, release until end and inverse AR-envelope: time until/from minimum value		
t_attack	**float4**	*attack time of AR-envelope*
t_release	**float4**	*release time of AR-envelope*
t_invattack	**float4**	*attack time of inverse AR-envelope*
t_invrelease	**float4**	*release time of inverse AR-envelope*
ADSR-envelope approximation		
t_A_time	**float4**	*attack time*
t_A_level	**float4**	*attack level*
t_D_time	**float4**	*decay time*

t_S_level	**float4**	*sustain level*
t_S_time	**float4**	*sustain time*
t_R_time	**float4**	*release time*
Spectral characteristics of feature curve (see section 11.3)		
s_mean	**float4**	*spectral mean*
s_std	**float4**	*spectral std*
s_skewness	**float4**	*spectral skewness*
s_kurtosis	**float4**	*spectral kurtosis*
Five spectral bands of feature curve. The band limits in Hz are: 0, 1, 10, 20, 40, 100		
s_band0	**float4**	
s_band1	**float4**	
s_band2	**float4**	
s_band3	**float4**	
s_band4	**float4**	

A.4.3 View UnitData

This view performs the join of UnitFeature (A.4.1) and CharacteristicValues (A.4.2) and thus normalises the unit data of static descriptors, providing the necessary default values for their characteristic values.

A.4.4 View CorpusUnitData

This view is optimised for selecting units from a specific category or corpus. It performs the join of UnitFeature (A.4.1) and CharacteristicValues (A.4.2) and thus normalises the unit data of static descriptors, providing the necessary default values for their characteristic values.

For instance,

```
SELECT mean, std FROM CorpusUnitData WHERE cid = 111 and ftid = 10
```

gets all data for corpus 111 and descriptor 10, or as

```
SELECT DISTINCT ON (uid, ftid) mean, std FROM CorpusUnitData
WHERE cid = 111 AND ftid = 10 ORDER BY uid, ftid, aid DESC
```

which gets the newest unit data which was most recently analysed (the one with the highest aid).

A.4.5 View BasefileUnitData

This view is optimised for selecting units from a specific sound file. It performs the join of UnitFeature (A.4.1) and CharacteristicValues (A.4.2) and thus normalises the unit data of static descriptors, providing the necessary default values for their characteristic values.

Use it like

```
SELECT mean, std FROM BasefileUnitData WHERE bfid = 47 and ftid = 20
```

to get all data for basefile 47 and descriptor 20, or as

```
SELECT DISTINCT ON (uid, ftid) mean, std FROM BasefileUnitData
WHERE cid = 47 AND ftid = 20 ORDER BY uid, ftid, aid DESC
```

to get the newest unit data which was last analysed (the one with the highest aid).

Appendix B

Database Interface (dbi) Reference

B.1 Startup and Utilities

help, fullhelp , lookfor

```
show quick help / full help for dbi subcommands
```

init, initdbi

```
initialize a connection with a database (caterpillar by default)
```

deinit, deinitdbi, close

```
close the connection with the current database
```

verbosity

```
set verbosity of psql functions (0-9)
```

loadsymbols

```
load symbols into struct (name -> syid)
```

begin

```
start high-level transaction
```

commit

```
end high-level transaction
```

rollback

```
undo high-level transaction
```

B.2 Categories and Corpora

addcategory, addcorpus

```
This function adds a category/corpus in the database and, if one or
more parents are given as following arguments, puts it under these
parent categories.  If no parents are given, a new root category is
created.

dbi addcategory name 'description' [parent-categories...]
```

deletecorpus, deletecategory, deletefeaturetype

```
delete category
(leaves connected categories, just removes connections)
```

deletecategorytree, deletecorpustree

```
delete category and all child-categories (dangerous!)
```

addcorpusparents

```
dbi('addcorpusparents', child, parents...)
add corpus as child to parents
```

opencorpus

```
open a corpus or a category by creating a join table of
caracteristics containing those of the segments included in this
category/corpus : cunit features and categories they are included in.
```

addtocategory, addtocorpus

```
Add a unit, several units, or a basefile (given by name or path only!)
in first argument to categories or corpora given in following arguments
(id or name).
```

removefromc, removefromcorpus, removefromcategory

```
Remove a unit or a basefile from a corpus or a category.
argument 1 is the unit (id) or a basefile (if it is a string)
argument 2 is corpus (id or name),
```

listcorpus

```
called without argument, this function lists the corpus (name,
ftid, number of basefiles associated) existing in the base
called with an argument from integer type, it returns the name,
ftid, number of basefiles associated with the corresponding identifier
called with an argument from  type string , it returns the name,
ftid, number of 'principal' units associated with the corresponding
corpus
```

getcorpusunits

```
returns the ids of units in given corpus, all units when empty
```

getcorpusregions

```
returns the identifiers of main units which
belongs to a corpus or a category
if empty argument is given, consider the current corpus
```

getcorpusfiles

```
getcorpusfiles returns the identifiers of BaseFiles
which belong to a corpus or a category and its children
```

listcategory

```
called without argument, this function lists the categories (name,
ftid, number of basefiles associated) existing in the base
called with an argument from integer type, it returns the name,
ftid, number of basefiles associated with the corresponding identifier
called with an argument from  type string , it returns the name,
ftid, number of 'principal'units associated with the corresponding category
```

isin

```
query category membership of a unit
dbi isin uid category-name
```

getcorpusstat

```
[ mean, std ] =  dbi('getcorpusstat', cid, ftid, xval)
return precomputed statistics over all units of a corpus, or the
whole database if cid is empty
returns two (1, N) row vectors where N = length(ftid) = length(xval)
```

B.3 Basefiles

getbasefile

```
Gives the characteristics (id, name, path, extension) of a basefile (or
several ones), given by name or by id as arguments
called without arguments, it returns the caracteristics of all the
basefiles of the database
```

getbasefileid

```
get bfid of base file given by name
```

getsoundfileid

```
get bfid of sound file given by name
```

getbasefilestruct

```
return basefile record as struct array
```

getsoundfilestruct

```
return basefile record as struct array
```

verifybasefiles

```
check if the attribute 'path' of concerned basefiles (list given by
the user, or all ) references a real file .
return paths of inexistant basefiles
```

addbasefile

```
add a basefile to the database, add the corresponding unit, add to
categories if given
```

addvirtualfile

```
add a virtual file and the units it contains to the database,
```

deletebasefile, deletesoundfile

```
delete a sound file and all the references on it in the database:
Units, unit data, feature files, analysis runs
```

deletebasefileunits

```
delete a basefile's units and their data
```

getbasefilemainunit

```
get unit encompassing the whole basefile, created with it
basefile can be given as bfid, path, or name.
```

B.4 Feature Files

addfeaturefile

```
Add a feature file to the FeatureFile table and to the AnalysisRun
table. This makes sense, because we don't want feature files to pop
up freely just like that.
the three first arguments : BaseFile id, FeatureAnalysis ids, path of
the new FeatureFile are necessary
the subformat (4th argument) is optional.
```

addanalysisrun,

```
Create a new analysis run entry from a basefile, a featurefile
(possibly NULL), and a feature analysis

aid = dbi('addanalysisrun', bfid, ffid, faid)
```

getanalysisrun,

```
Query analysis run entry for a basefile, a featurefile
and a feature analysis, all given by name or id.

[aid, faid, bfid] = dbi('getanalysisrun', bfname, ffname, faname)
```

getrawdata

```
bpf = dbi('getrawdata', ftid, aid, unit) get raw continuous feature data
of type ftid from feature file analysed in aid.

Unit is either empty, then get the whole data,
a single integer, then get data for that uid.
or a 2-element array, then it is the extent of the data in seconds

(Raw data access is handled by the dbi, too, since we need it
both for unit feature calculation, and for displaying by dbx.)
```

B.5 Feature Types and Analysis

getfeaturetype

```
get normalized id of feature type
```

listfeaturetypes

```
get a list of feature types and categories which are not root
(not corpus)
```

listfeatureanalysis

```
get a list of feature analysis
```

getanalysisfeaturetypes

```
get the featuretypes an analysis returns
```

addfeaturetype

```
add a new feature type

dbi('addfeaturetype', 'name', 'unit', 'datatype', 'comment')

datatype is one of 'real', 'int', 'bool', 'symbol', 'text'
```

addfeatureanalysis

```
add a new feature analysis entry

dbi('addfeatureanalysis', 'featureanalysis-name', 'comment')
dbi('addfeatureanalysis', 'featureanalysis-name',
'version', 'args',  'comment', [sdifsel])

with sdifsel = list of pairs of featuretype name and sdifselection, e.g.
dbi addfeatureanalysis Ecrins 'Ecrins descriptor analysis' f0 1FQ0 energy 1NRG
which will be added to table Analyses
```

addanalyses

```
add the fact that a featureanalysis generates a featuretype

dbi('addanalyses', faid, [ ftid ])
dbi('addanalyses', faid, 'featuretype1', 'sdifselection1', ...)
```

getfeatureanalysis

```
return faid according to component name (todo: compversion, etc...)
```

B.6 Units, Unit Data, and Characteristic Values

addunit

```
Add a unit.  Parameters are the basefile associated, the start and
end time of the segment.
Basefile is automatically added as parent, as are the unit's
UnitFeature entries for type, start, end, duration.
```

getbasefileunits

```
Return the id, start time, end time, and duration of the units of a
given basefile (or several), with optional
preselection conditions in the following args
The first unit is the first file's main unit
```

getbasefileunitseq

```
Return the units of a given basefile (or several) as structure, with
optional condition in args 2, 3.  Order is by uid.
```

getunitstruct

```
Return the unit as struct in the given sequence, duplicates
not removed
```

getunitseq

```
Return the unit entries as struct in the given sequence, duplicates
not removed
```

getunitbasefile

```
Return the basefile id of the given list of units
```

listcharacteristics

```
return list of characteristic value names
```

getbasefilefeatures

```
Return all ftids present for all the units of a basefile
```

getunitdata

```
data = dbi('getunitdata', selector, id, ftid, chval, aid,
[unittypes], [preselection])
returns the data for units from tables
UnitFeature and CharacteristicValues directly.

Arguments:
uid vector(N) of unit ids.
ftid vector(M) of descriptor ids,
chval string cell(M) of characteristic value fields.
If only one string is given, use it for all ftid.

Result:
data (N, M) matrix of unit data, one column per 'feature'
(defined by a (ftid, chval) pair)
```

getunitdatastruct

```
data = dbi('getunitdatastruct', selector, id, ftid, aid)

get complete CharacteristicValues as (N, M) struct array data
for given unit ids (N) and feature ids (M) vectors

WARNING!  Each unit data structure takes up around 3800 bytes.
A small corpus of 5000 units takes then 18 MB per feature!
```

addunitdata

```
dbi('addunitdata', uid, aid, ftid, unitdata)
insert unitfeature or unitdata = feature+characteristicvalue
(it depends on arguments) for a unit
```

getsymbols

```
get symbols
```

B.7 Miscellaneous queries for inspection and maintenance

vacuum

Routine maintenance of database: reclaim spece of unused tuples, update the table statistics, and reorder UnitData (A.4.3) for the most frequent access.

getrelations

childrenof

parentsof

showisa

maketree

`Draw category tree under given categories, all by default`

maketreein

`Draw category tree, write to given dir`

treepic

`Generate tree and show it using xv and GhostView`

catree

Appendix C

Database Explorer (dbx) Reference

help

```
Show quick help for dbi subcommands
```

fullhelp

```
Show full help for dbx subcommands
```

tables, db

```
Show database tables
```

soundfiles, sounds

```
Show all soundfiles in database, select one to display
```

corpus, files

```
Show all soundfiles in database, ordered by corpus tree,
select one to display
```

basefiles, allfiles

```
Show all files (sound and descriptor) in database, select one to
display
```

plotbasefile, plotsoundfile

```
Plot one soundfile and the current descriptor
```

plotfeaturefile,

```
Plot one descriptor data file
```

highlightunit,

```
Synchronise views:  Highlight the same unit in all.
```

setfeature,

```
Display this feature in unitzoom
```

syncview,

```
Synchronize time-range of all time-dependent views
```

synclegend,

```
Switch legend on or off in all views
```

zoomunit

```
dbx('zoomunit', uid, uidset) -- display unit and its current ftid data:
characteristic values and raw data
```

explore

```
Main unit explorer, arg is corpus, none means all units
```

Appendix D

Documentation of the Documentation Scripts

D.1 doccase

The *doccase* script generated the documentation of the *dbi* and *dbx* switchboard functions in appendices B and C. It is also used to provide interactive help for these function, being called by `dbi help` and similar. Its help message is:

```
Usage:
        doccase [options] file.m [topic]

Options:
        -h     this help
        -v     verbose
        -k     search for keyword in topic
        -quick quick help
        -latex output help in LaTeX

Parse a matlab switchboard function in the given file.m for 'case'
statements, which represent subfunctions of the main function,
extract comments starting with the documentation marker '%%', and
print a formatted list of case labels (the subfunctions) with
their documentation.

If -quick is given, only show the case label and the first line of the
documentation.

If a topic is given, give full help only for this topic, or if -k is
given, print subfunctions matching the string in topic.
```

D.2 docsql

The *docsql* script is used to generated the documentation of the CATERPILLAR SQL schema in appendix A, and the UML diagrams in figures 14.5 and 14.4.

It parses the table creation statements in the SQL source file for table name and attribute definitions, key declarations and REFERENCES statements, and FROM clauses for views.

For the table documentation, it generates LaTeX code, for the UML diagrams, it writes a graph description file in the syntax of the automatic graph layout program DOT[1] (Koutsofios and North 1996), which then finds the best disposition for the edges and nodes.

Its help message is:

```
Usage:
        docsql [options] file.sql

Options:
        -h       this help
        -v       verbose
        -uml     output UML diagrams
        -noviews don't output views
        -max     output longest names of tables, attributes, and datatypes

Parse a file with SQL table creation statements, extract comments, and
output the schema with one of the three following handlers:

LaTeX (the default)

    Write LaTeX code to stdout using a set of macros to build the
    tables with attributes and description, and headings.

-uml

   Generate a UML diagram in the form of a source file for the "dot"
   automatic graph layout program.  Foreign key references and view
   dependencies are extracted as edges of the graph, adorned by the
   name of the referencing attribute.

-max

   Write short LaTeX file with a macro definition of the longest name
   of all tables, attributes, and datatypes


The comment syntax for LaTeX is the following: Only comments starting
with the documentation marker '---' are treated.  Only tables or views
having documentation are considered for output.

A block of comments before a table or view definition becomes the
table documentation.  A comment block before an attribute definition
is the description of this attribute.

Groups of attributes, or a heading within the attributes can be
defined by '--!'.  The same documentation marker outside of a table
creates a top-level heading in the LaTeX output.

Each table starts a normal-level heading, and comments starting with
'--2' start a sub-level heading.  For views, a line with '--^',
followed by a list of table names forces these tables to be used as
the dependencies of the view, and not the automatically parsed list
after the FROM clause (which can contain complex sub-SELECT
expressions).
```

[1] http://www.graphviz.org

Appendix E

SDIF Description Types

The Sound Description Interchange Format (SDIF) described in (Virolle 1998; Wright, Chaudhary, Freed, Khoury, and Wessel 1999) is used for well-defined interchange of data with external programs (analysis, segmentation, synthesis). We give here the details of the different description types used in CATERPILLAR. For an introduction to the principles and advantages of SDIF, see section 13.4.1.

Frame and matrix types are defined in an XML file, from which the official description type documentation[1] is generated. Here, we use the somewhat cryptic but more concise syntax of the SdifTypes.STYP file, that is also generated from the XML file:

Matrix type definition:

```
1MTD   matrix-signature { column1, column2, ... }
```

Frame type definition:

```
1FTD   frame-signature
       {
          matrix-signature1   NameOfThisMatrixInThisFrameType;
          matrix-signature2   AnotherMatrix;
       }
```

The convention for signature classes is: normal signatures start with a number, giving the version of that type. As the types are well thought out, this is always `1`. Matrices with additional information about parameters that were used to compute the actual data, start with `I`. Non-standard matrices, i.e. that are experimental or not bound to become part of the standard start with `X`.

E.1 SDIF Types for Segments

General marker frame type, output from segmentation programs (directly or via a translator) It carries the *segmentation confidence* and optionally a label.

We also write the aligned score events into the segmentation file, to keep the data together for visualisation and evaluation. For this, a description type `1MID` was defined that codes the midi file verbatim into SDIF. A `1MID` Midi frame contains either a `1MID` matrix with the raw Midi events making up the aligned chord in its rows, or alternatively a `1SYX` matrix with system exclusive data.

```
1MTD  1SEG    { Confidence   }
1MTD  1LAB    { TextOrNumber }
```

[1] http://www.ircam.fr/sdif/standard/types-main.html

```
1FTD  1SEG
      {
        1SEG  Segmentation;
        1LAB  Label;
      }

1MTD  1MID    { Status, Data1, Data2 }
1MTD  1SYX    { Data }

1FTD  1MID
      {
        1MID  MIDIEvent;
        1SYX  MIDISystemExclusive;
      }
```

E.2 SDIF Types for Descriptors

These SDIF types allow to export the sound descriptor from the analysis programs, and to import them into the Caterpillar database. See chapter 10 for meanings of the types.

```
1MTD  ICEN    { Scale }
1MTD  1CEN    { Centroid }
1MTD  1LDN    { Loudness }
1MTD  1NPS    { Ratio }
1MTD  IPAR    { EvenPartAmplNum, OddPartAmplNum, EvenPartAmplDen, OddPartAmplDen }
1MTD  1PAR    { Parity }
1MTD  IRGL    { Methods }
1MTD  1RGL    { Slope, Intercept, SquareMeanError }
1MTD  1SHP    { Sharpness }
1MTD  ISKE    { Scale }
1MTD  1SKE    { Skewness }
1MTD  ISPR    { Scale }
1MTD  1SPR    { Spread }
1MTD  ISRG    { Methods }
1MTD  1SRG    { Slope, Intercept, SquareMeanError }
1MTD  ITRI    { FirstPartialNum, LastPartialDen }
1MTD  1TRI    { Tristimulus }
1MTD  1TWD    { TimbralWidth }
1MTD  XHMC    { Harmonicity }

1FTD  1LDN
      {
        1LDN  Loudness;
      }
1FTD  1PAR
      {
        IPAR  Info;
        1PAR  Parity;
      }
1FTD  1RGL
      {
        IRGL  Info;
        1RGL  Coefficient;
      }
```

```
1FTD  1SKE
      {
        ISKE  Info;
        1SKE  Skewness;
      }
1FTD  1NPS
      {
        1NPS  Coefficient;
      }
1FTD  1SHP
      {
        1SHP  Sharpness;
      }
1FTD  1SRG
      {
        ISRG  Info;
        1SRG  Coefficient;
      }
1FTD  XHMC
      {
        XHMC  Harmonicity;
      }
1FTD  1TRI
      {
        ITRI  Info;
        1TRI  Tristimulus;
      }
1FTD  1SPR
      {
        ISPR  Info;
        1SPR  Spread;
      }
1FTD  1TWD
      {
        1TWD  TimbralWidth;
      }
1FTD  1CEN
      {
        ICEN  Info;
        1CEN  Centroid;
      }
```

E.3 SDIF Types for Semiphones

These phonetic descriptors are exported from the EULER MLC and imported into CATERPILLAR. See section 17.5.2 for more details.

```
1MTD  XWRD   { PhoneticWord }
1MTD  XSYL   { Syllable }
1MTD  XBEN   { BeginSemiPhoneFlag, EndSemiPhoneFlag }
1MTD  XDUR   { SemiPhoneDuration }
1MTD  XGRN   { GrammaticalNature }
1MTD  XFST   { FirstPhoneInSyllableFlag, FirstSyllableInWordFlag, FirstWordInSentenceFlag }
1MTD  XLEX   { LexicalWord }
1MTD  XPHO   { XSampaPhoneme }
```

```
1MTD  XLST    { LastPhoneInSyllableFlag, LastSyllableInWordFlag, LastWordInSentenceFlag }
1MTD  XPOS    { PositionInSyllable, PositionInWord, PositionInSentence }
1FTD  XSPH
      {
        XDUR  SemiPhoneDuration;
        XPOS  Positions;
        XPHO  XSampaPhoneme;
        XBEN  BeginOrEndSemiPhone;
        XSYL  Syllable;
        XWRD  PhoneticWord;
        XLEX  LexicalWord;
        XGRN  GrammaticalNature;
        XFST  FirstPhonemeFlags;
        XLST  LastPhonemeFlags;
      }
```

Appendix F

The X-SAMPA Computer Readable Phonetic Alphabet

X-SAMPA[1] (Wells 1995) encompasses the language-specific extensions of the SAMPA[2] (Speech Assessment Methods Phonetic Alphabet) phonetic alphabet, defined to make the standard IPA phonetic alphabet[3] (IPA 2003) usable on computers. The SAMPA alphabet for French[4] is explained in (Wells 2003), and copied here for convenience:

F.1 Consonants

The standard French consonant system is considered to consist of 12 obstruents (six plosives and six fricatives) and 8 sonorants (three nasals, two liquids, and three semivowel glides). The obstruents can be classified in voiced and voiceless pairs, with strong periodicity (voicing) normally occurring in the phonemically voiced members.

The plosives are p b t d k g:

Symbol	Word	Transcription
p	pont	po~
b	bon	bo~
t	temps	ta~
d	dans	da~
k	quand	ka~
g	gant	ga~

The voiceless plosives (/p t k/) are unaspirated except in stressed syllables preceding close vowels, where the extreme position of the tongue delays voice onset and produces turbulence. There are six fricatives, f v s z S Z; there is also j, which may be considered a fricative or a glide:

f	femme	fam
v	vent	va~
s	sans	sa~
z	zone	zon
S	champ	Sa~
Z	gens	Za~
j	ion	jo~

There are three nasals, m n J, found in words considered to be genuinely French. A fourth nasal, N, is only found in loanwords, except in Southern French dialects, where it ocurs in some contexts after nasal vowels:

[1] http://www.phon.ucl.ac.uk/home/sampa/x-sampa.htm
[2] http://www.phon.ucl.ac.uk/home/sampa/home.htm
[3] http://www.arts.gla.ac.uk/IPA/ipachart.html
[4] http://www.phon.ucl.ac.uk/home/sampa/french.htm

m	mont	mo~
n	nom	no~
J	oignon	oJo~
N	camping	ka~piN

There are two liquids, l R, and three vowel glides, w H and j. The vowel glides may be realised as fricative following voiceless obstruents.

l	long	lo~
R	rond	Ro~
w	coin	kwe~
H	juin	ZHe~
j	pierre	pjER

F.2 Vowels

The vowel system comprises 12 oral vowels, i e E a A O o u y 2 9 @, and 4 nasal vowels, e a o 9 , exemplified as follows:

i	si	si
e	ses	se
E	seize	sEz
a	patte	pat
A	pâte	pAt
O	comme	kOm
o	gros	gRo
u	doux	du
y	du	dy
2	deux	d2
9	neuf	n9f
@	justement	Zyst@ma~
e~	vin	ve~
a~	vent	va~
o~	bon	bo~
9~	brun	bR9~

When they are functional, the load of the oppositions a-A, e -9 , e-E, o-O, 2-9 may be very low for certain speakers, and there is a tendency towards neutralisation. When they are not functional there is a strong tendency in unstressed syllables towards indetermination. "Indeterminacy" symbols have been agreed to cover occurrences of these phonemes or sounds:

E/	= e or E
A/	= a or A
&/	= 2 or 9
O/	= o or O
U~/	= e~ or 9~

There are contextually determined vowel length differences, nasal vowels being long before following consonants, and all vowels being long before R and voiced fricatives.

Appendix G

Phonetic Categories

These category hierarchy trees generated from the database's IsA (A.1.3) relationship model the IPA phonetic classification (IPA 2003). The IPA symbols[1] making up the category names are coded in the X-SAMPA[2] computer readable phonetic alphabet (Wells 1995) (see also the previous section F and Bußmann 1990).

The unfortunately unreadable overviews still serve as orientation for the detailed subtrees following them.

[1] http://www.arts.gla.ac.uk/IPA/ipachart.html

[2] http://www.phon.ucl.ac.uk/home/sampa/x-sampa.htm

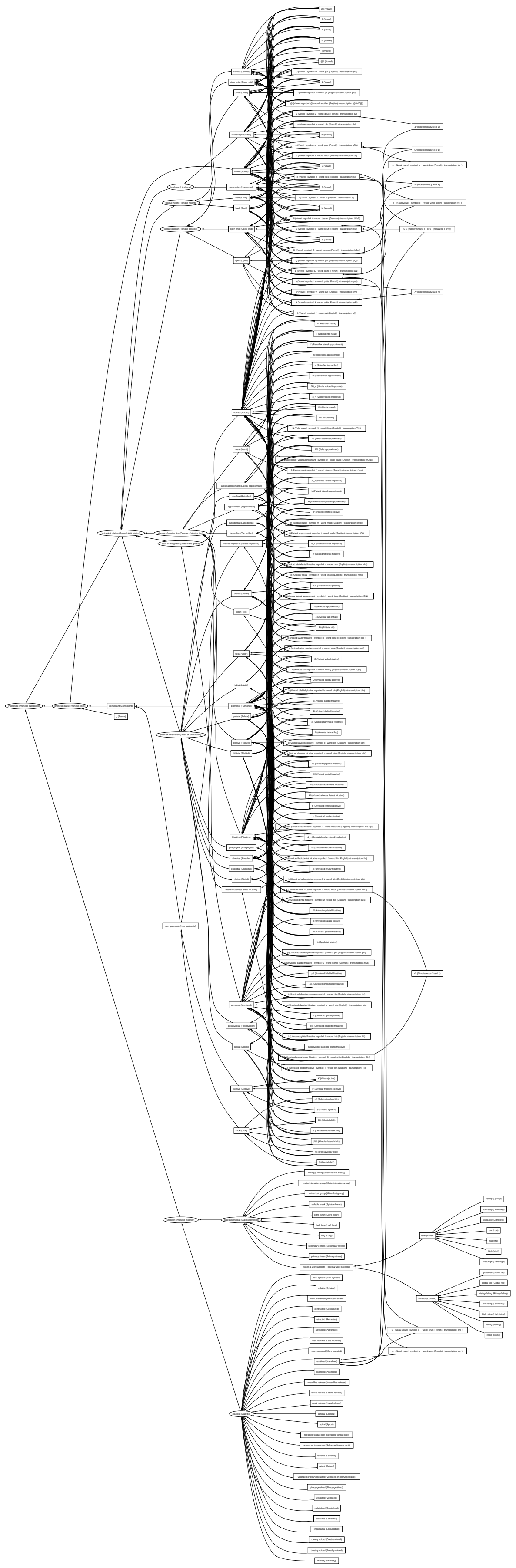


Figure G.1: Phonetic categories overview

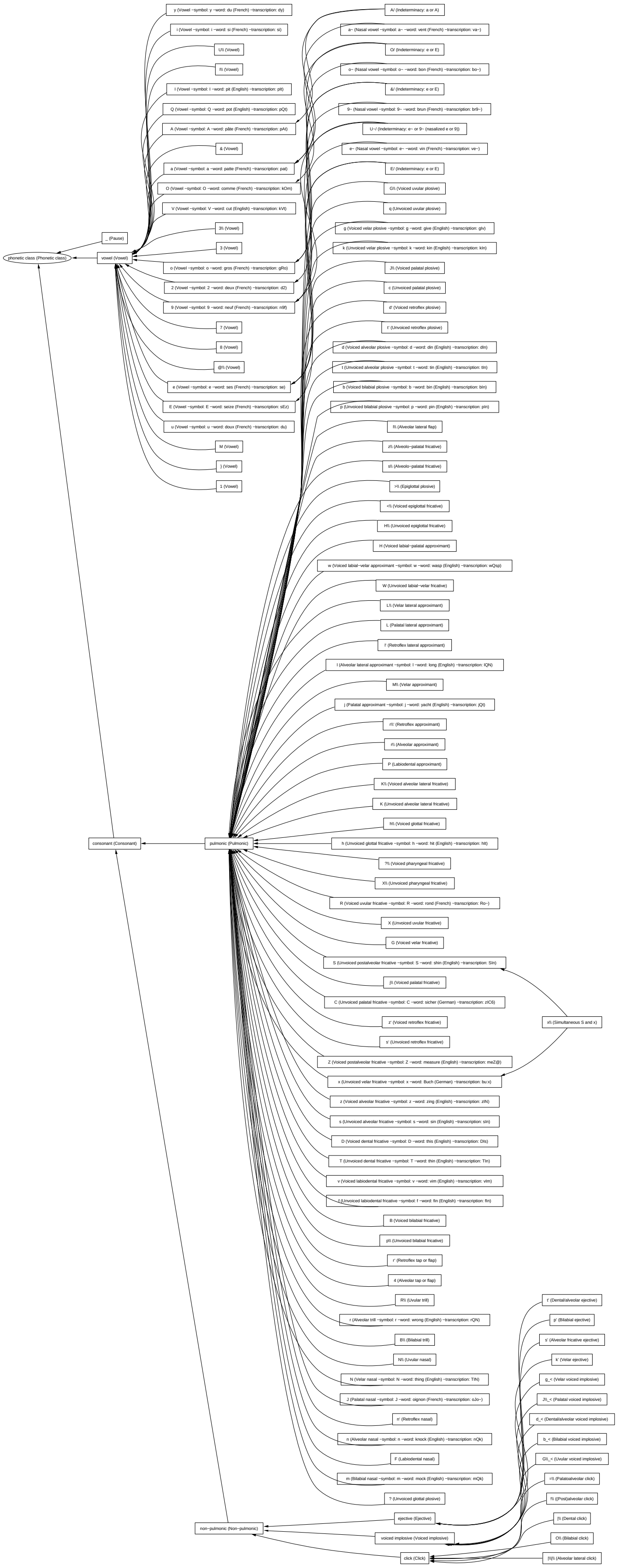


Figure G.2: Phoneme classes overview

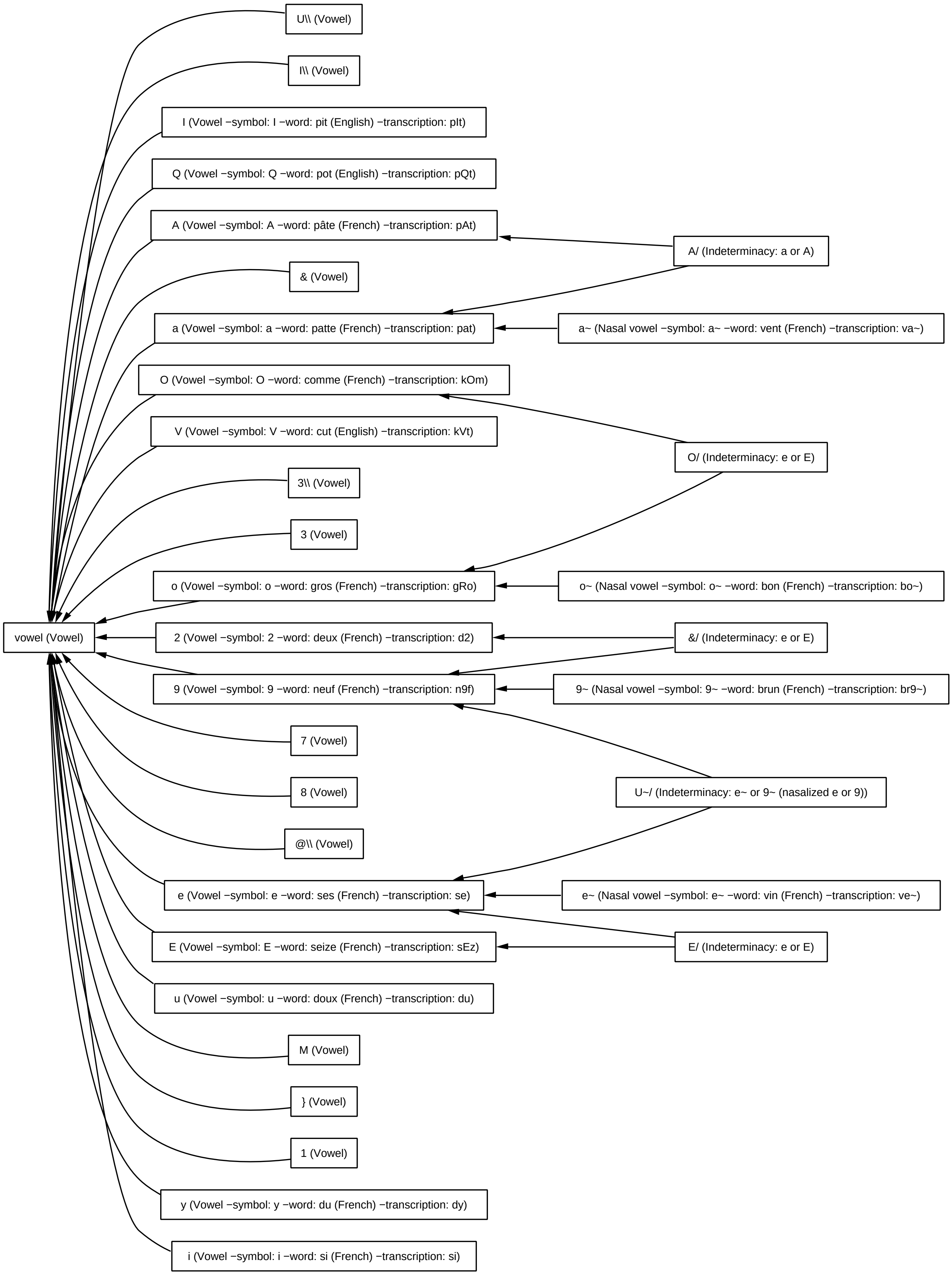


Figure G.3: Vowels

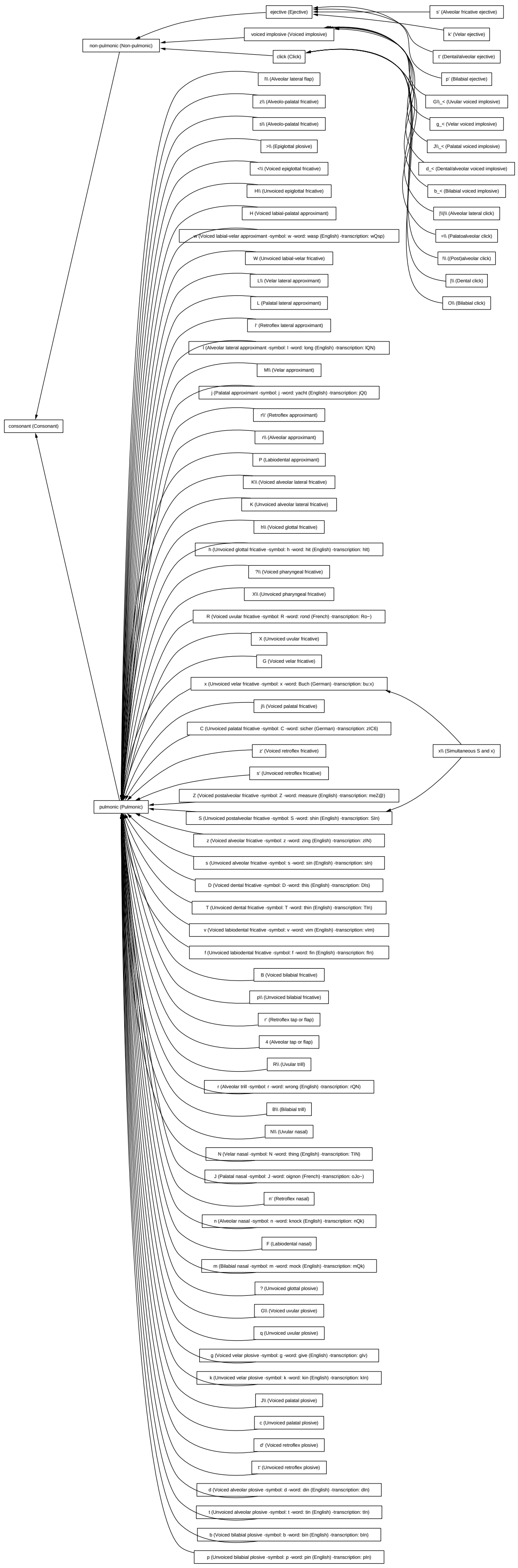


Figure G.4: Consonants overview

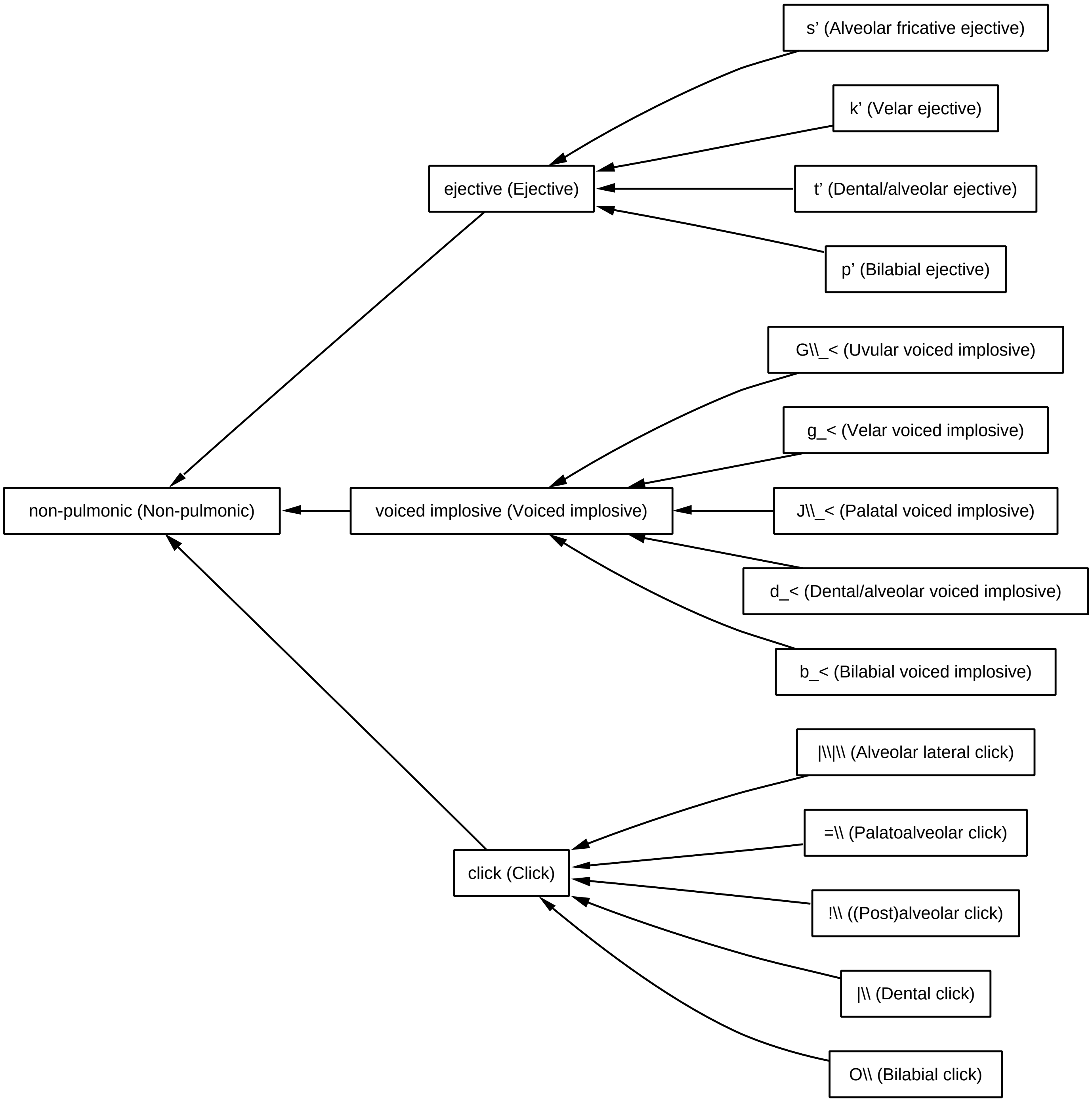


Figure G.5: Non-pulmonic consonants

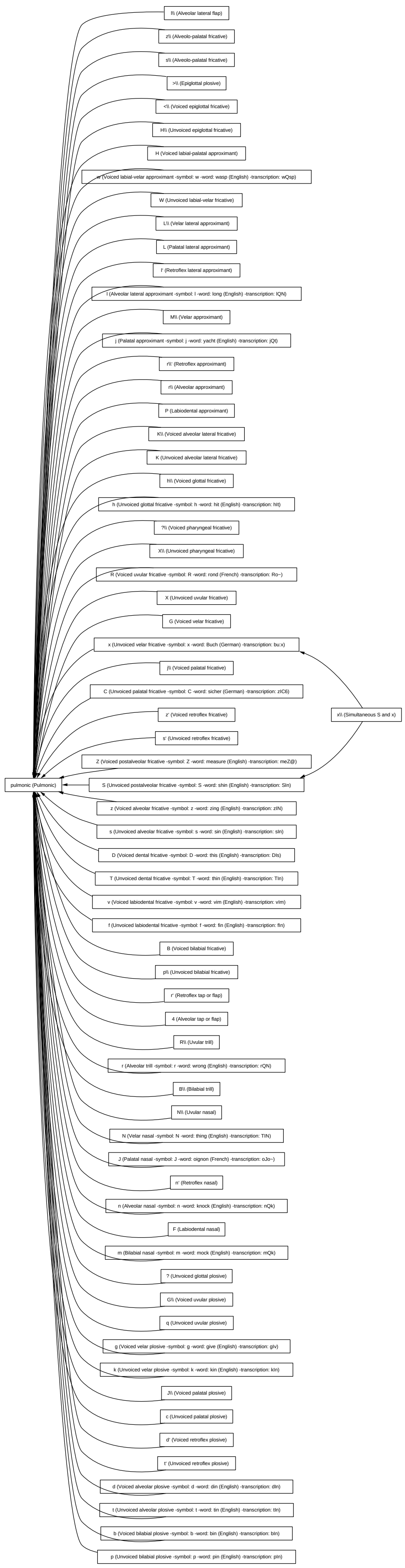


Figure G.6: Pulmonic consonants

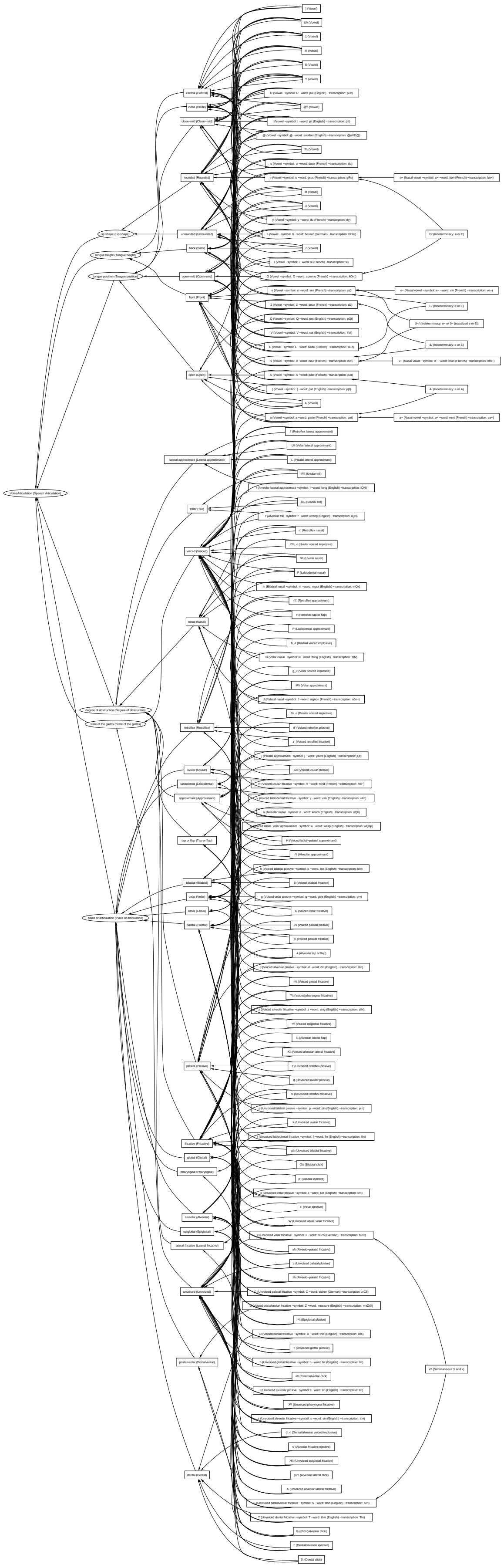


Figure G.7: Articulation overview

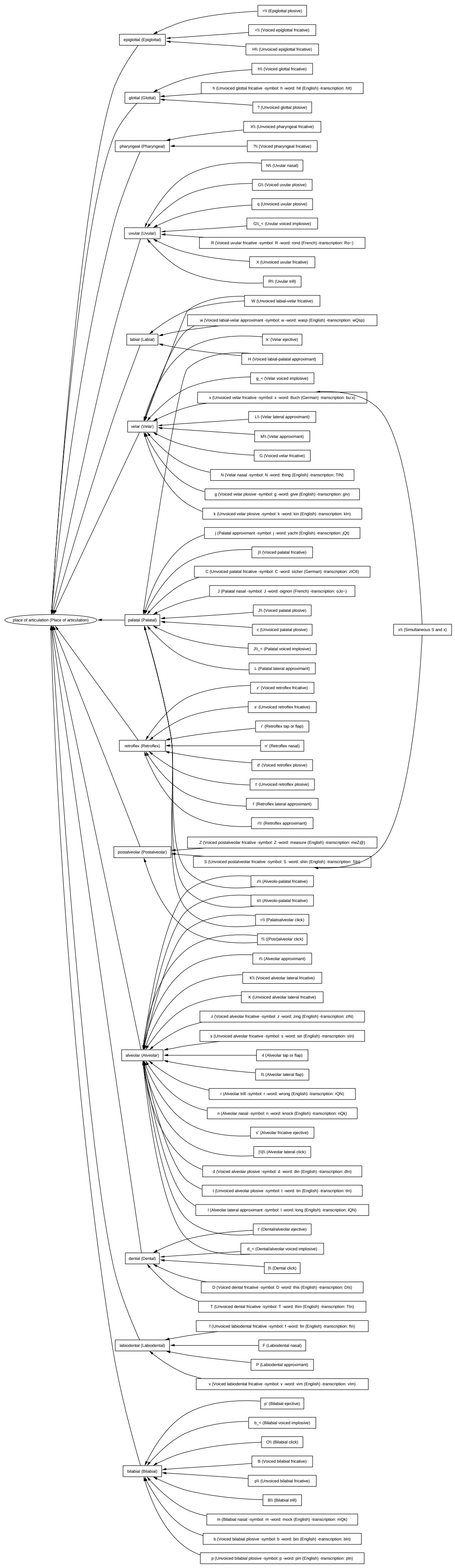


Figure G.8: Place of articulation overview

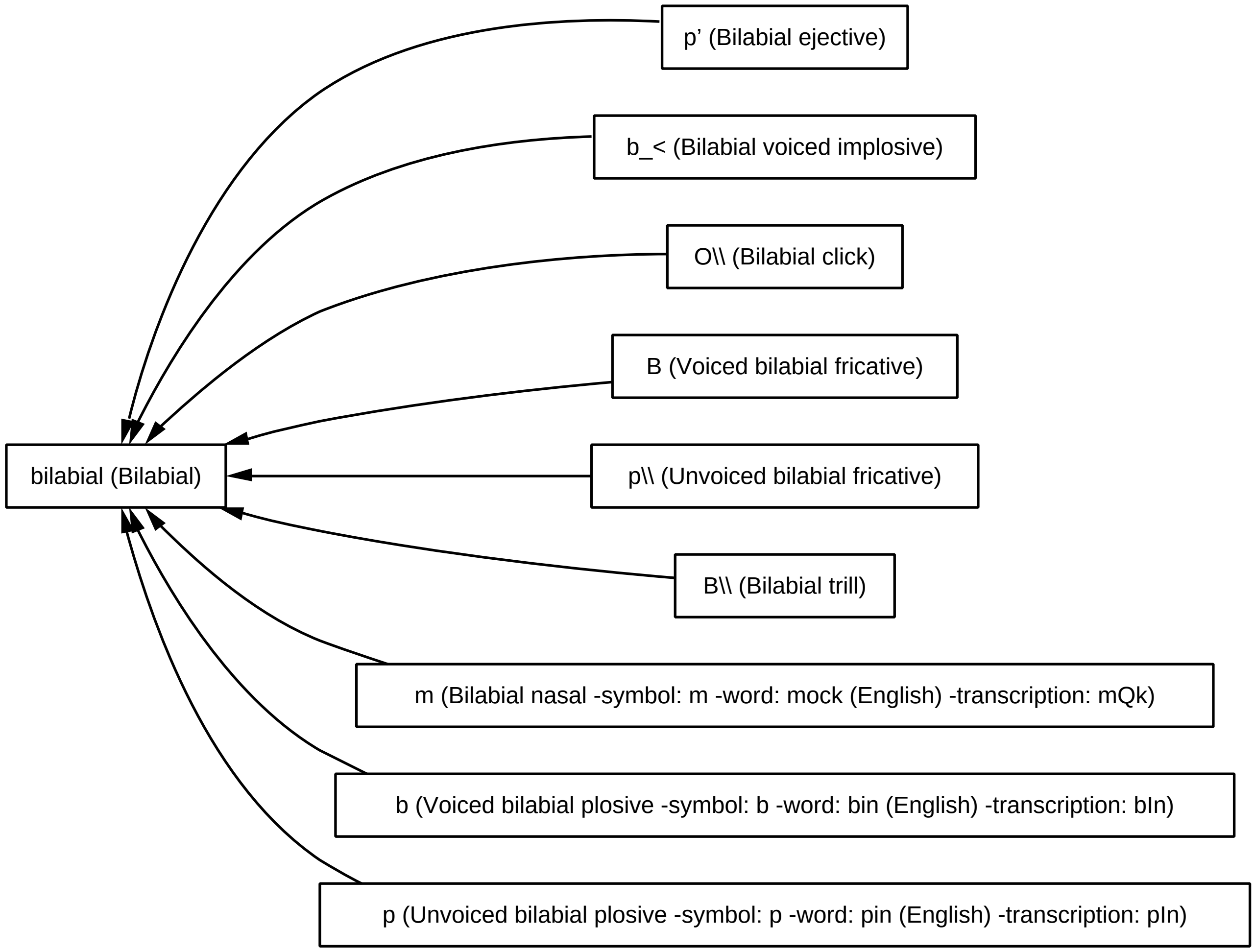


Figure G.9: Bilabial place of articulation

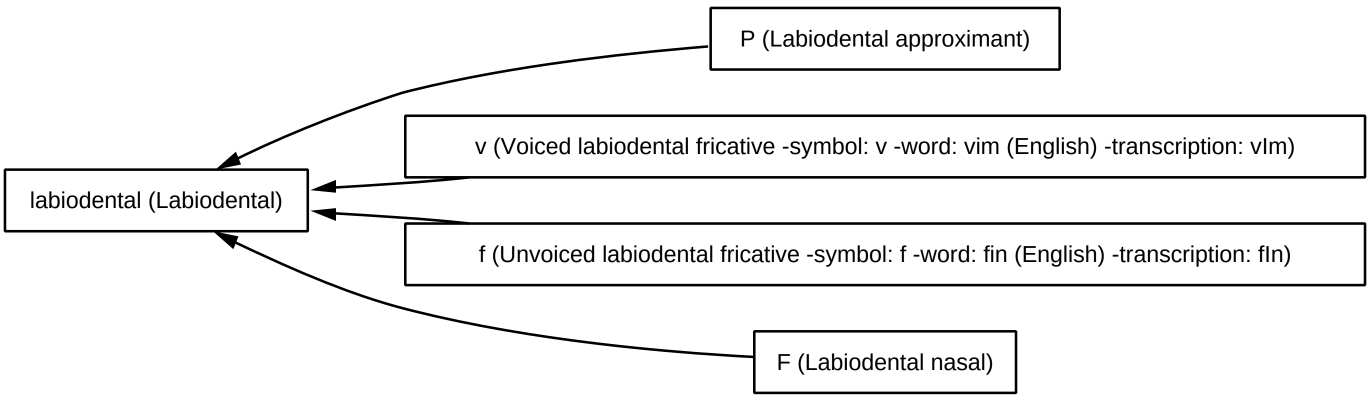


Figure G.10: Labiodental place of articulation

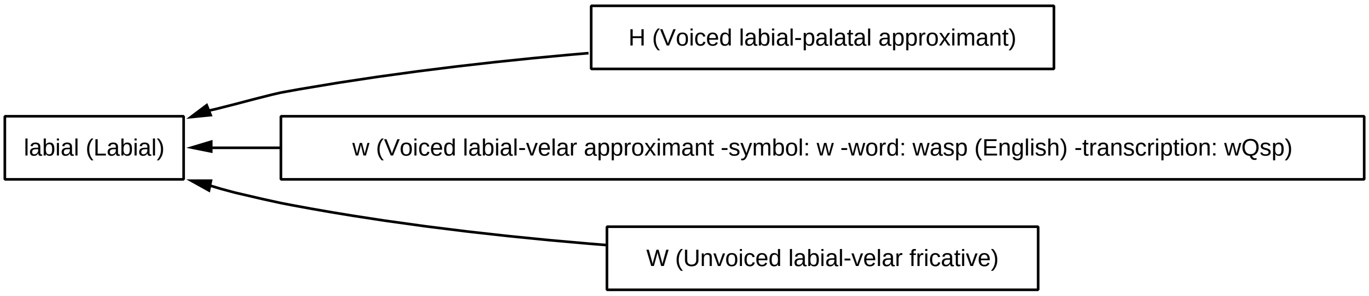


Figure G.11: Labial place of articulation

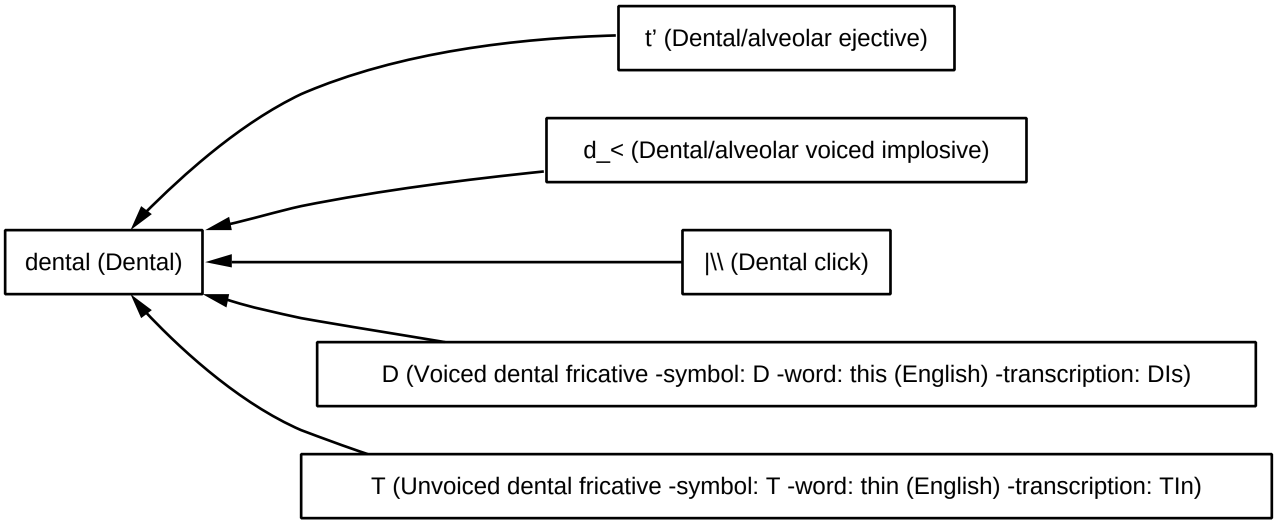


Figure G.12: Dental place of articulation

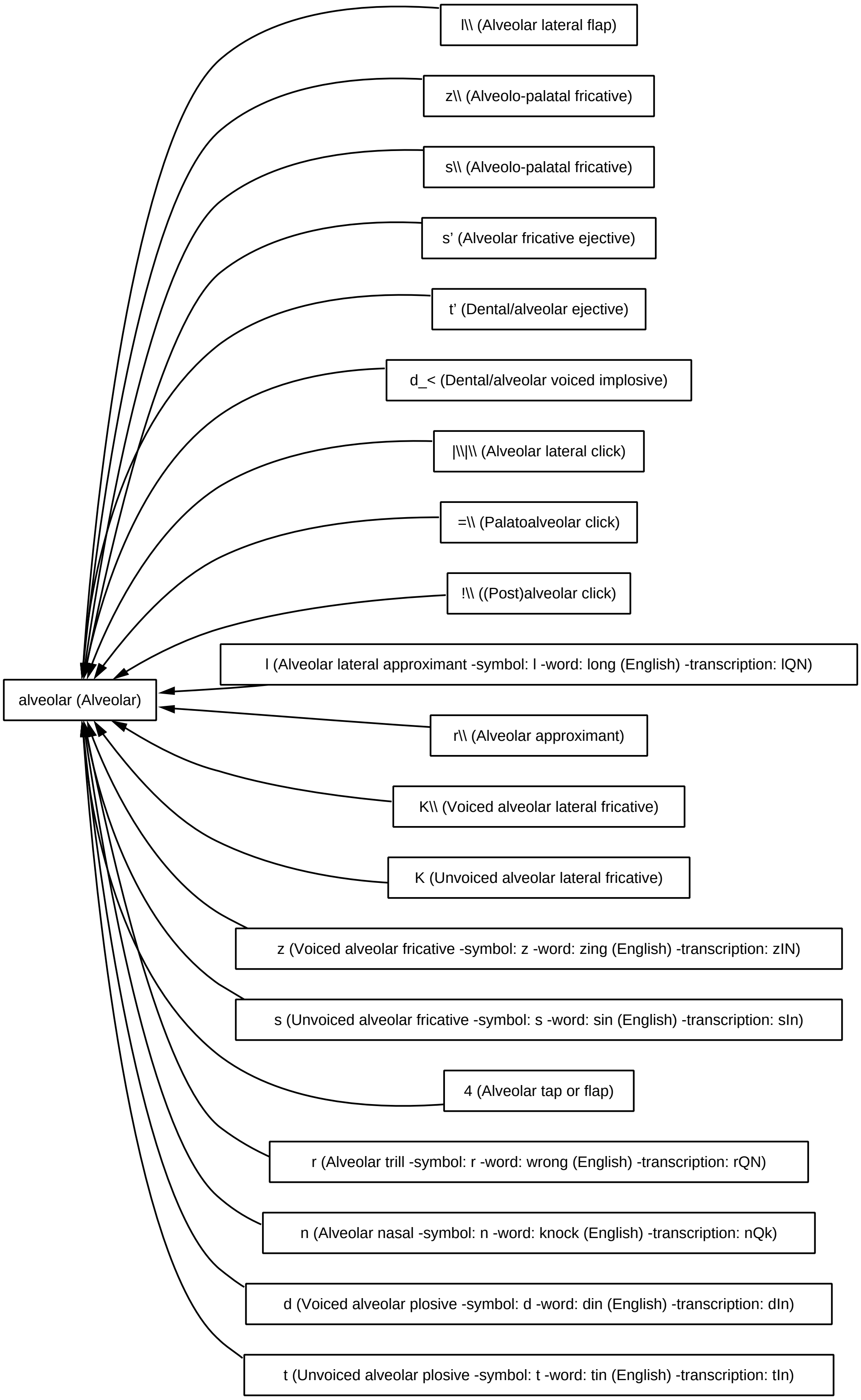


Figure G.13: Alveolar place of articulation

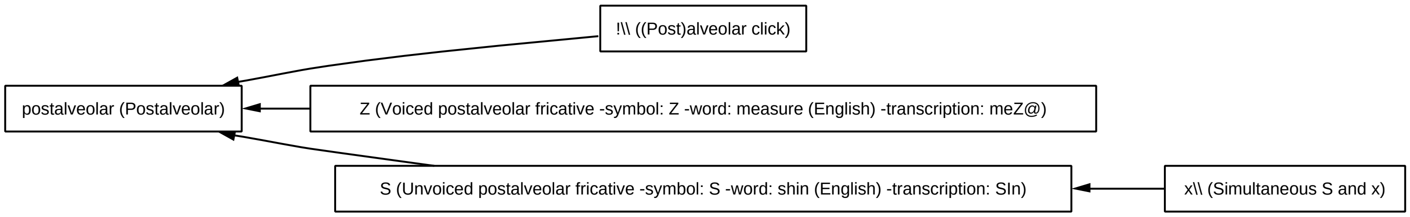


Figure G.14: Postalveolar place of articulation

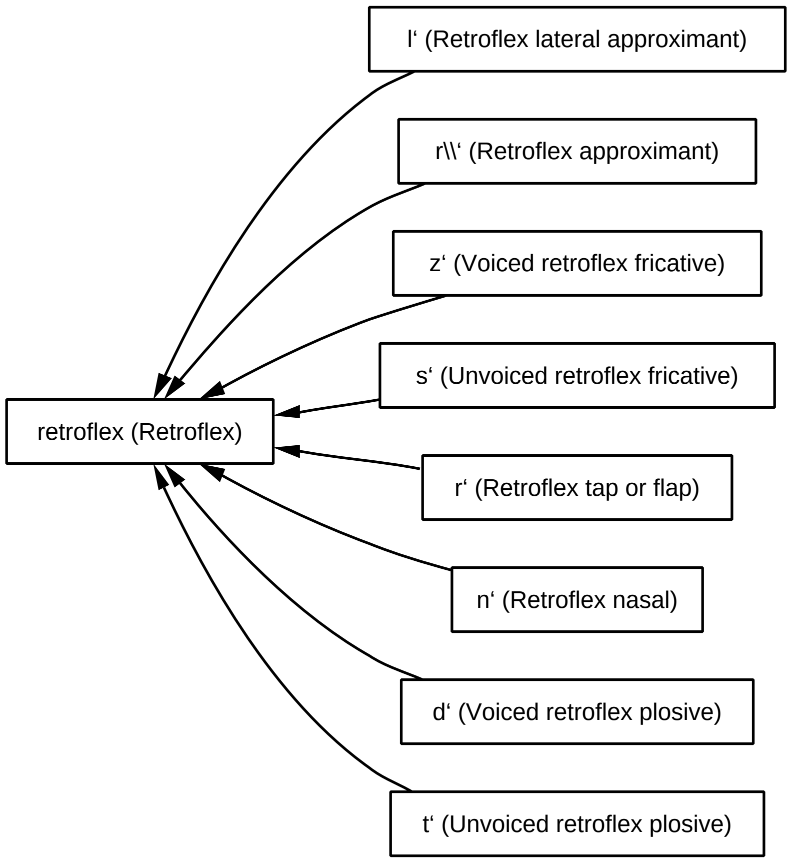


Figure G.15: Retroflex place of articulation

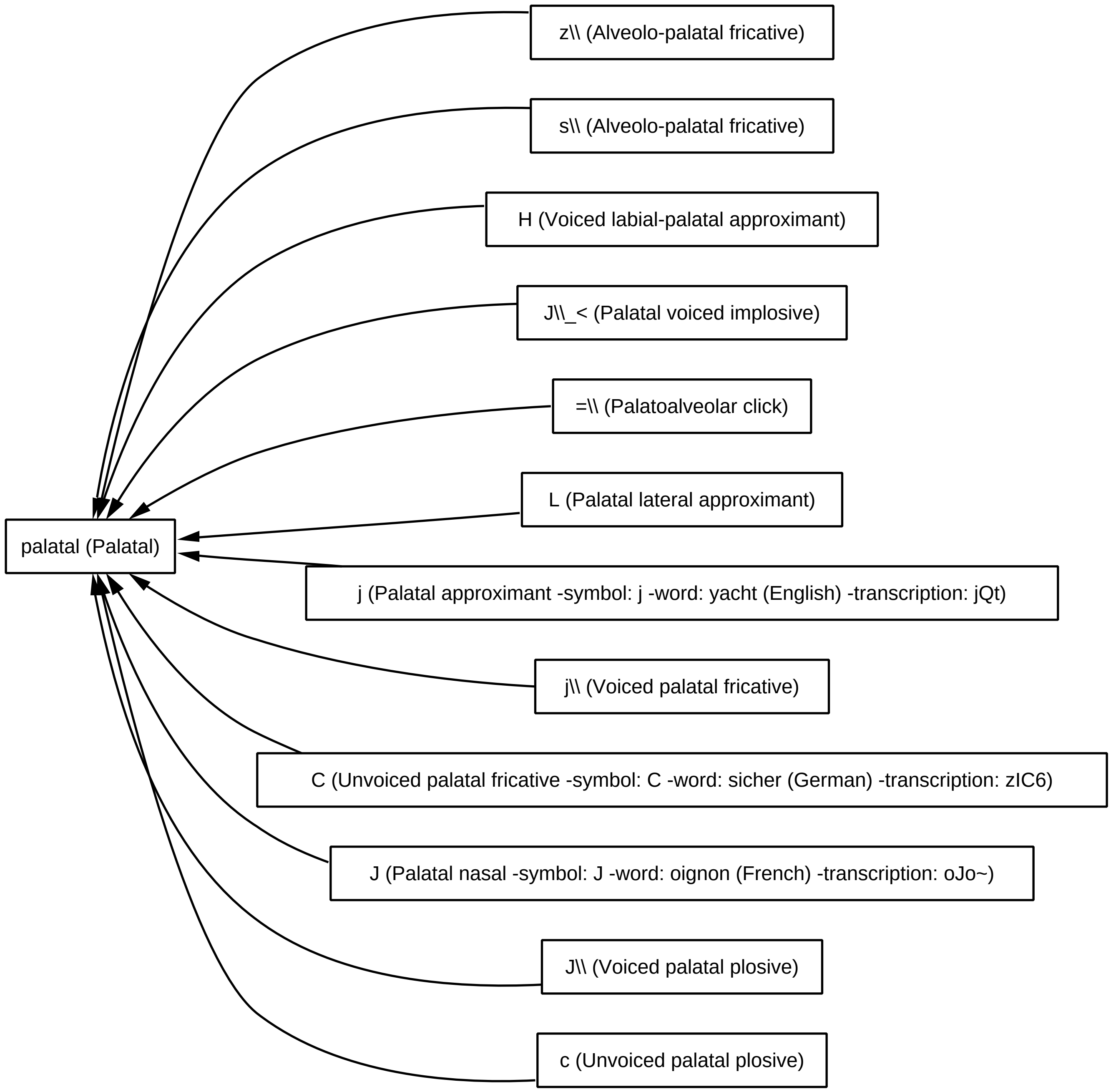


Figure G.16: Palatal place of articulation

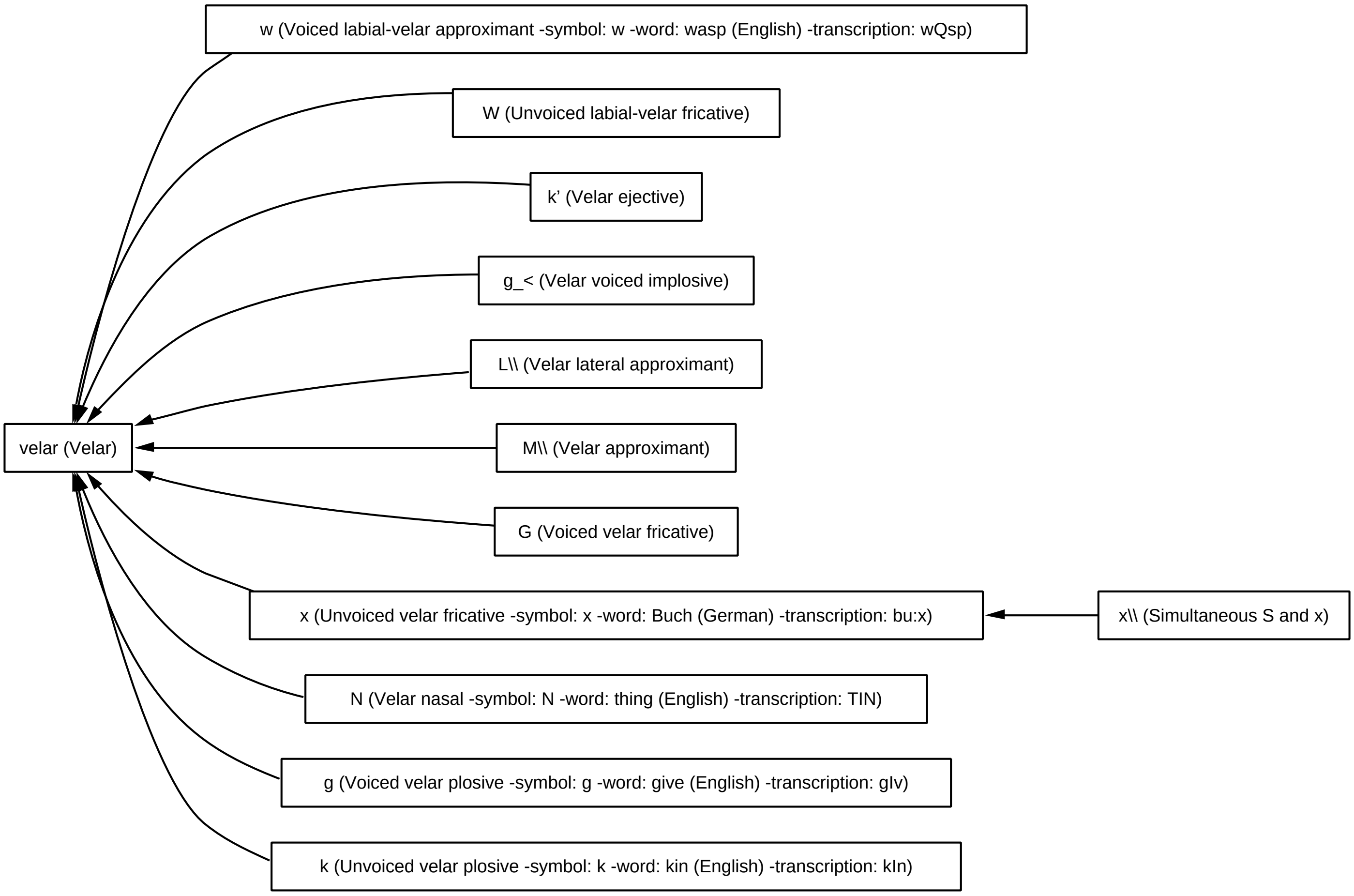


Figure G.17: Velar place of articulation

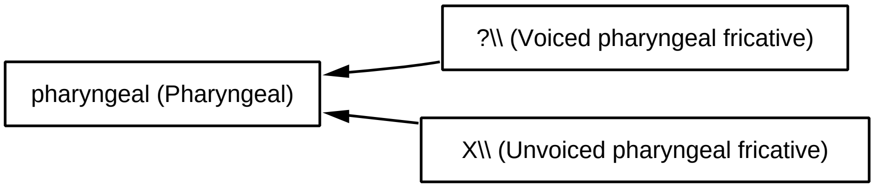


Figure G.18: Pharyngeal place of articulation

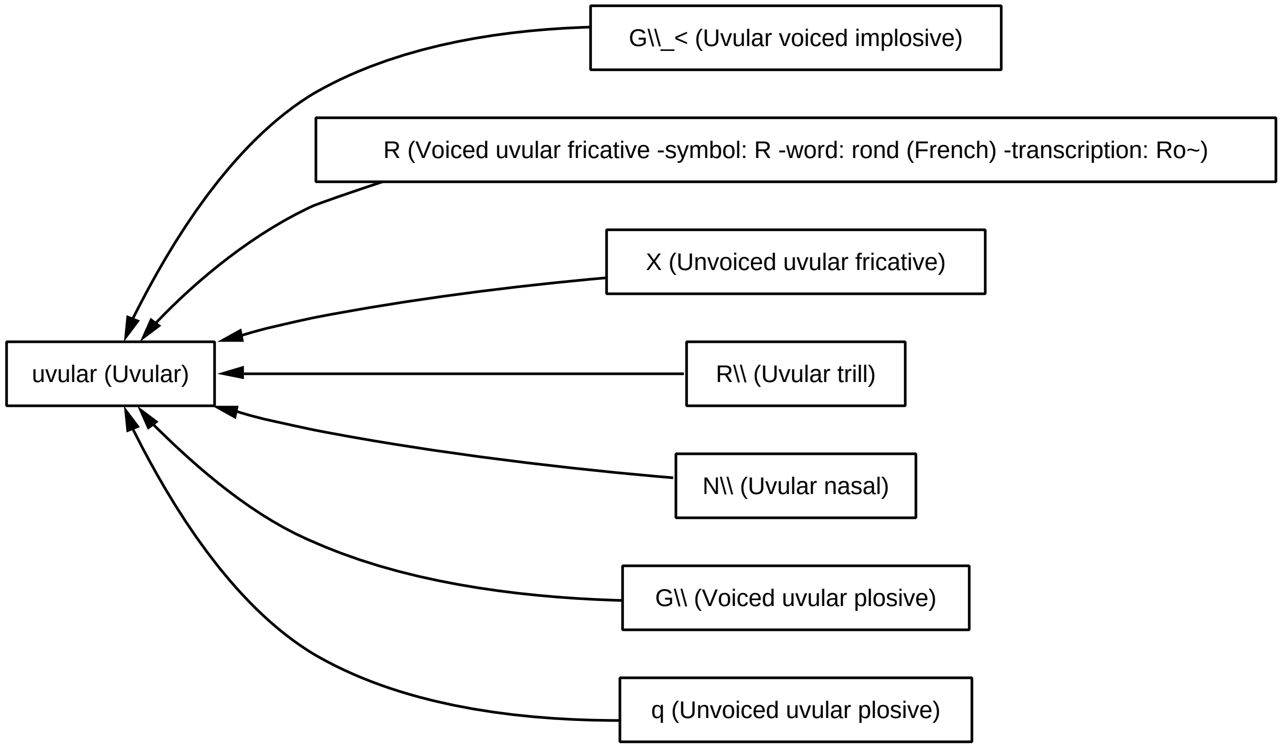


Figure G.19: Uvular place of articulation

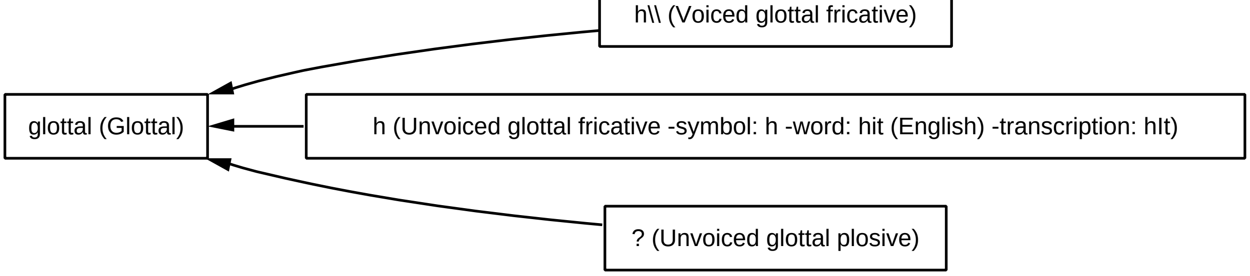


Figure G.20: Glottal place of articulation

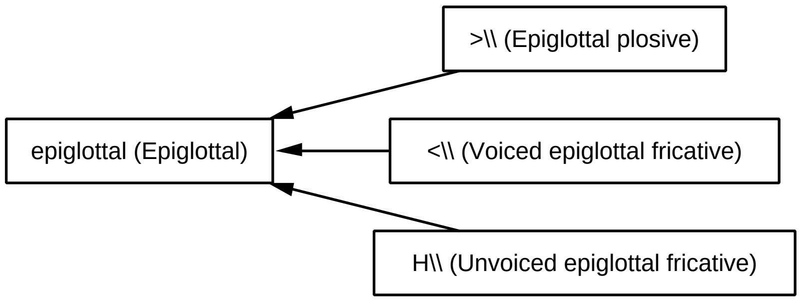


Figure G.21: Epiglottal place of articulation

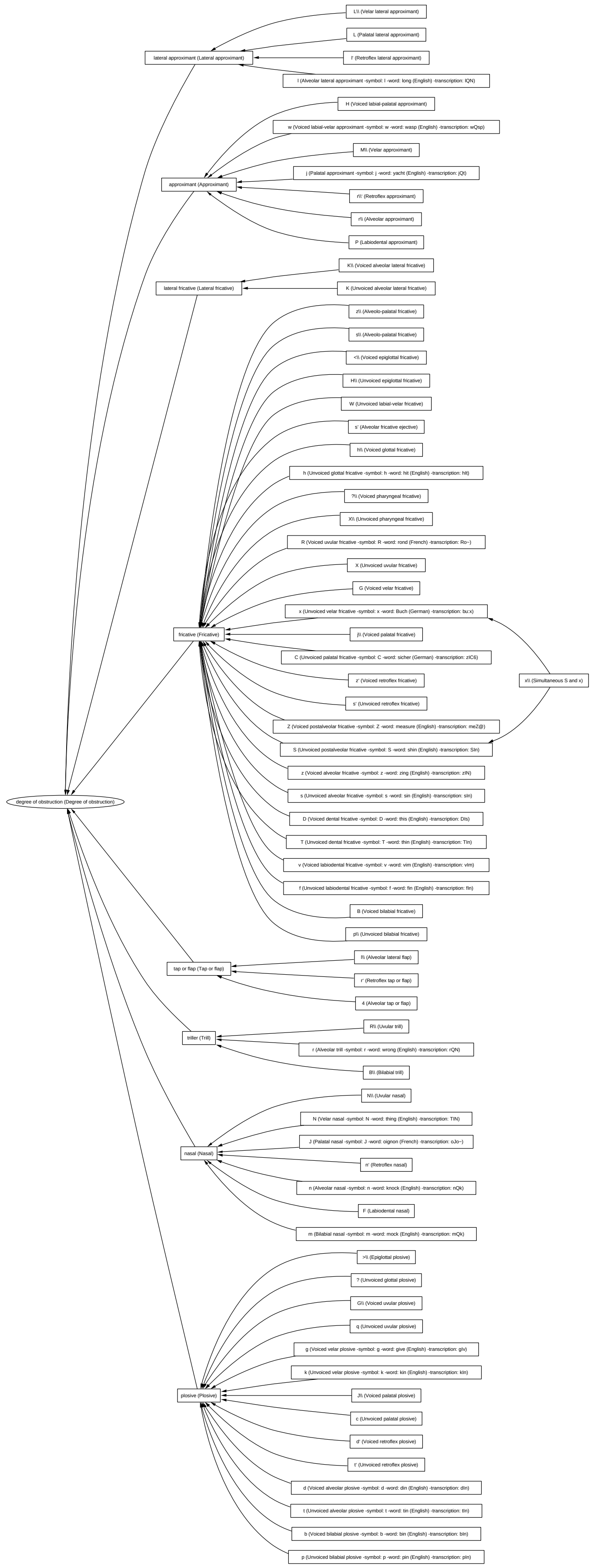


Figure G.22: Degree of obstruction overview

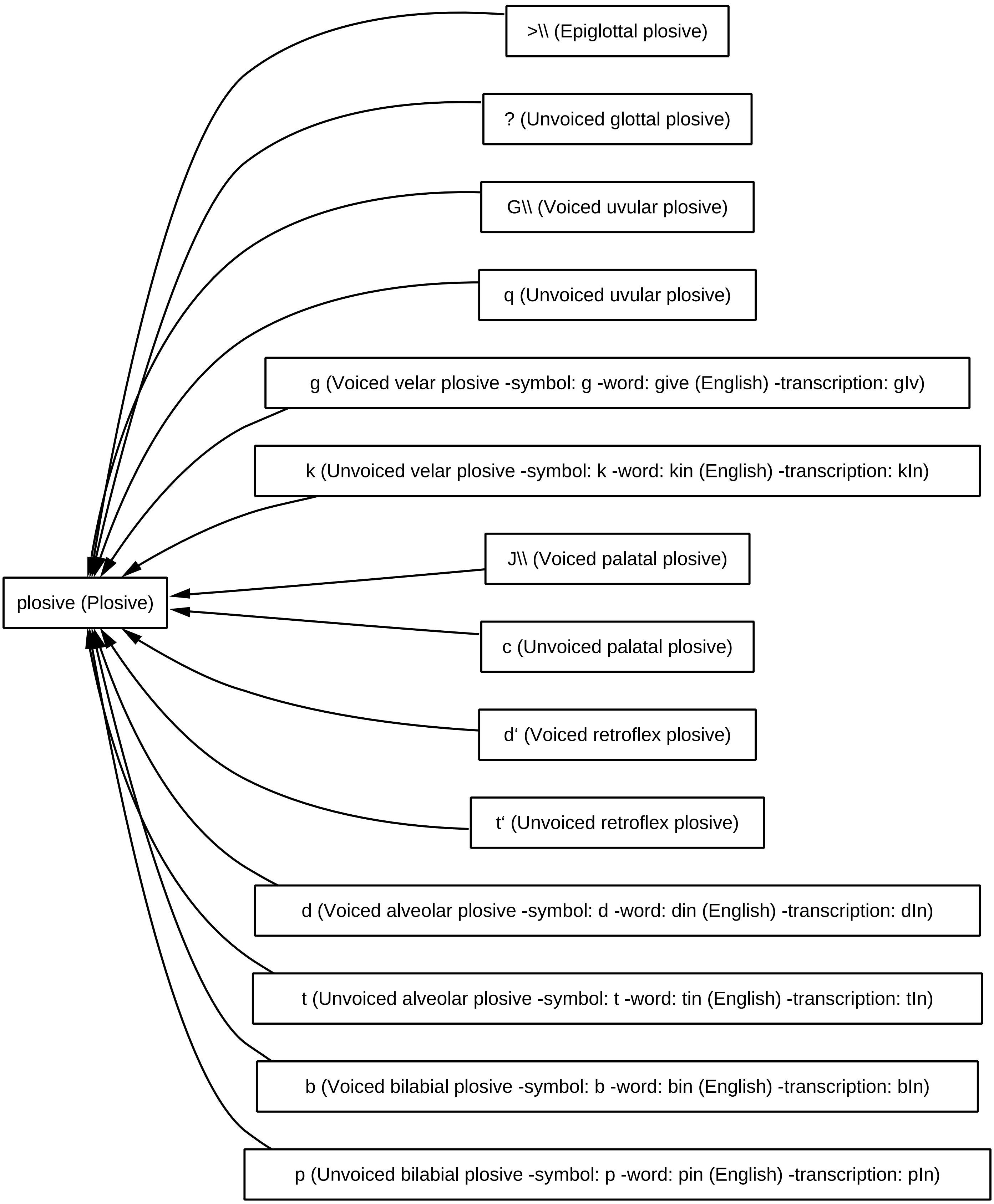


Figure G.23: Plosive obstruction

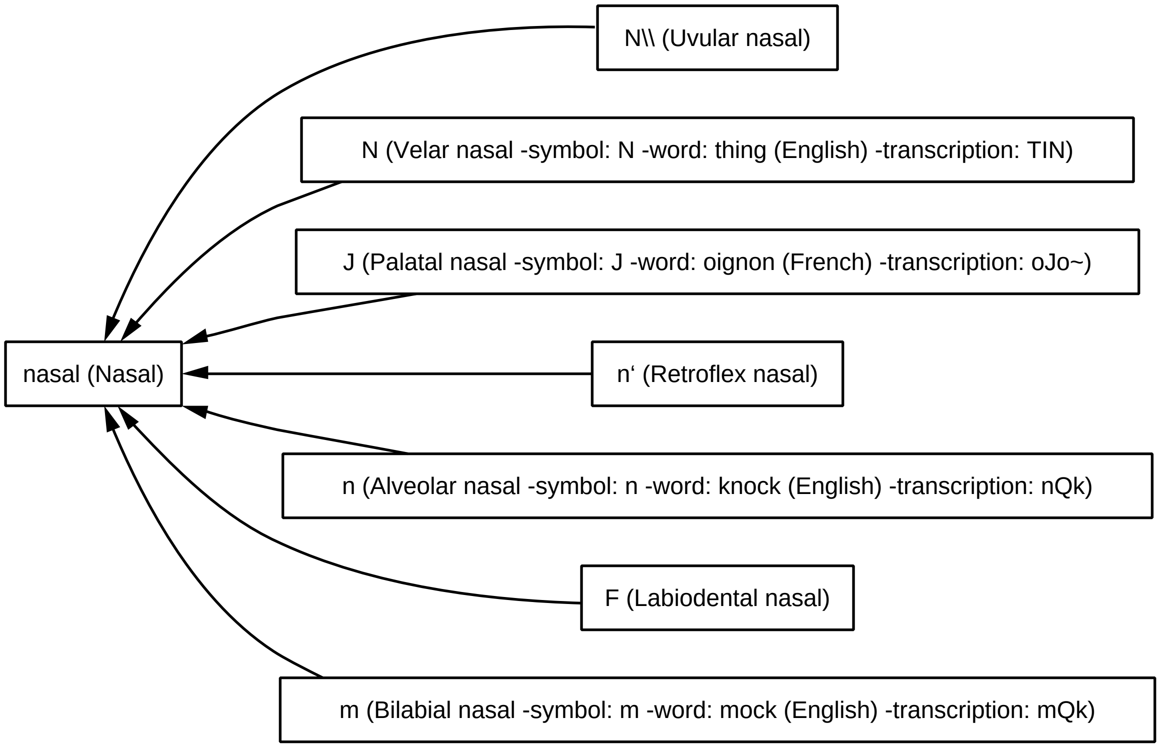


Figure G.24: Nasal obstruction

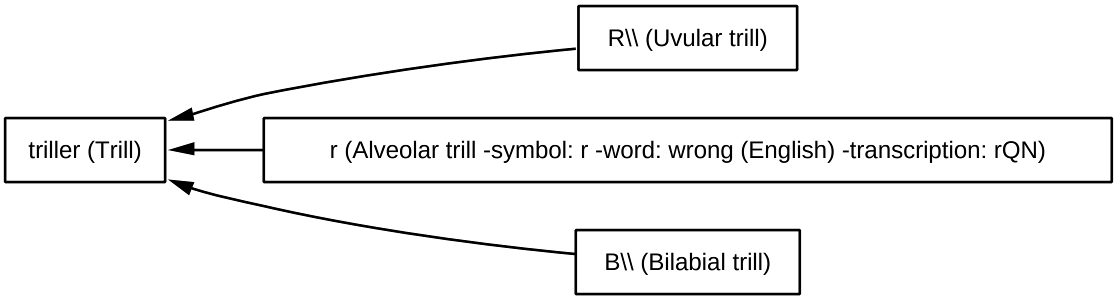


Figure G.25: Trill obstruction

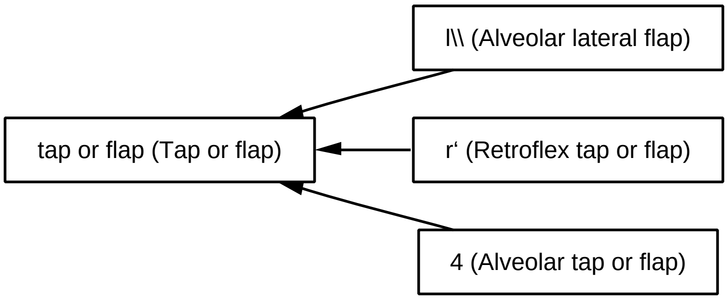


Figure G.26: Tap or flap obstruction

Figure G.27: Fricative obstruction

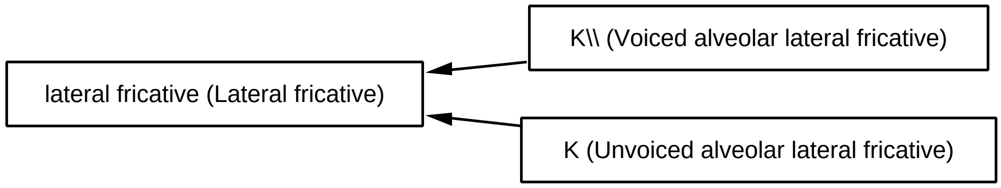


Figure G.28: Lateral fricative obstruction

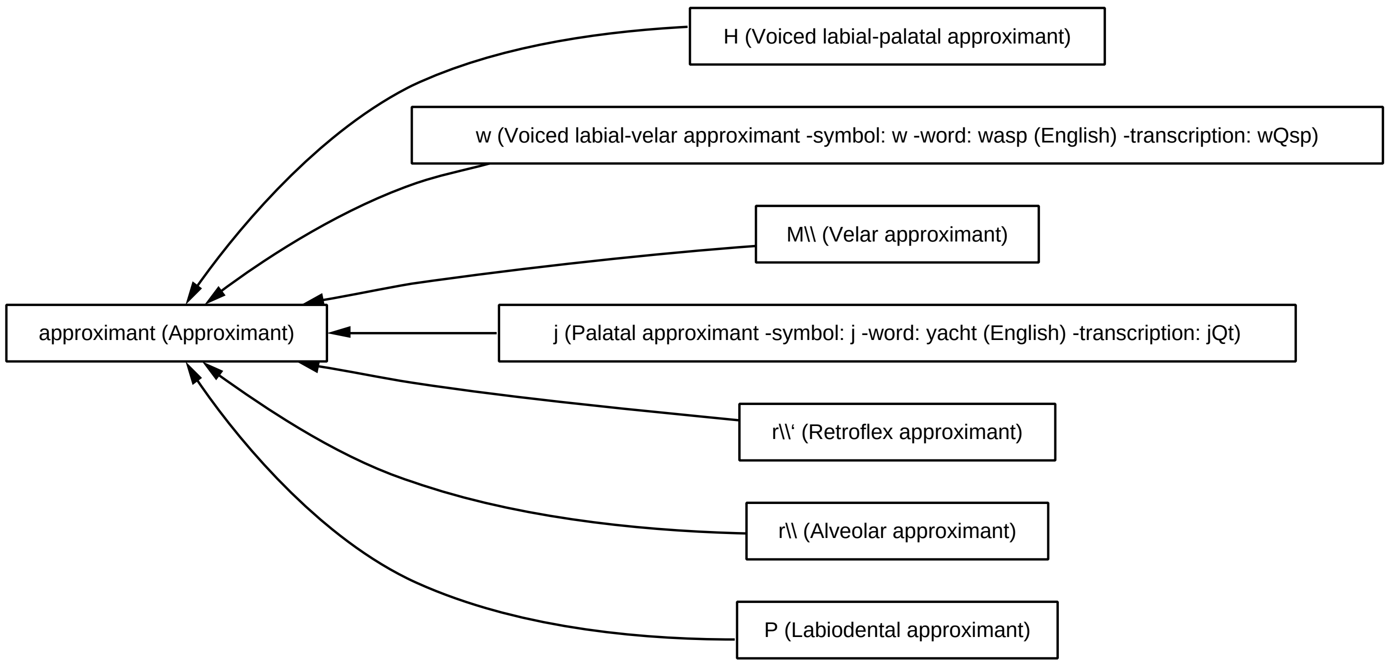


Figure G.29: Approximant obstruction

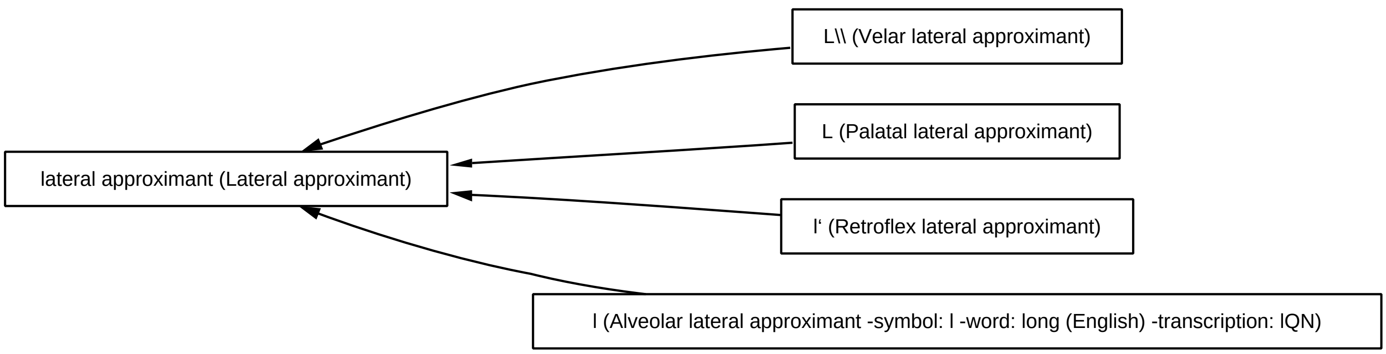


Figure G.30: Lateral approximant obstruction

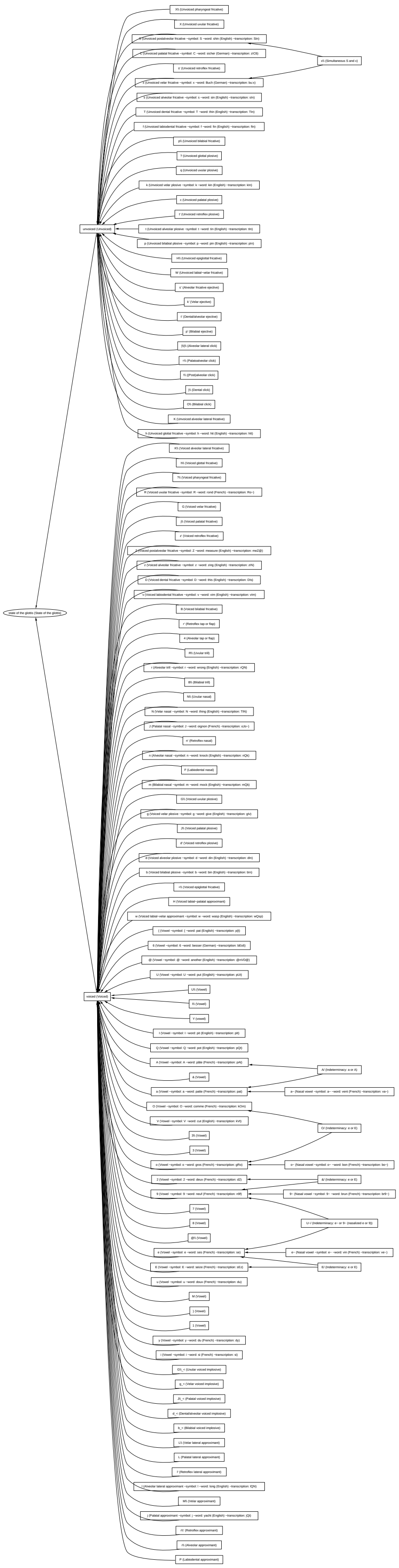


Figure G.31: State of the glottis overview

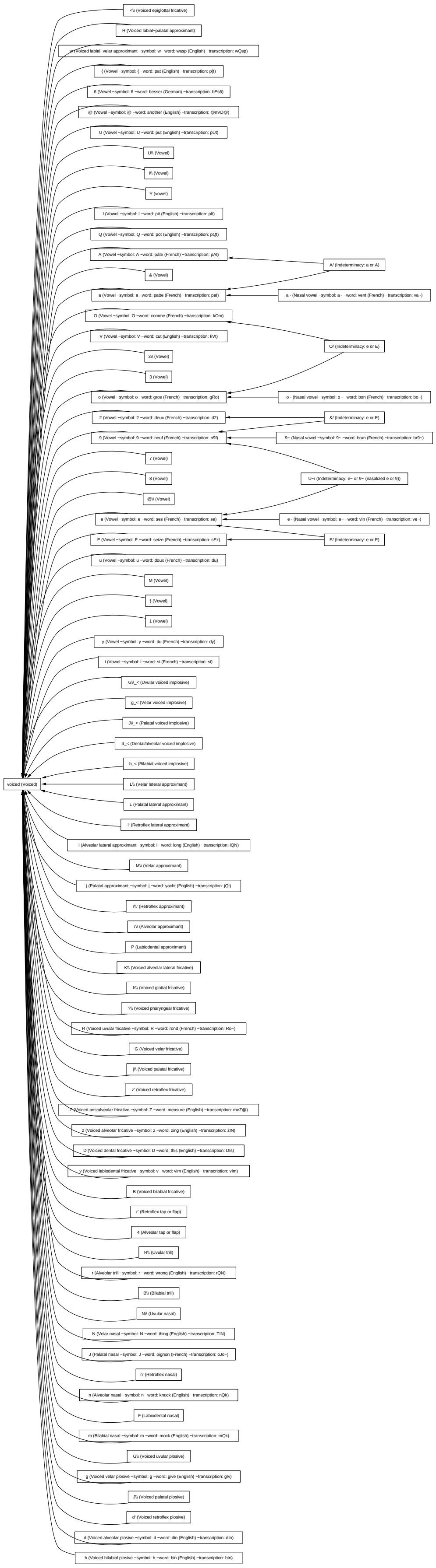


Figure G.32: Voiced glottis state

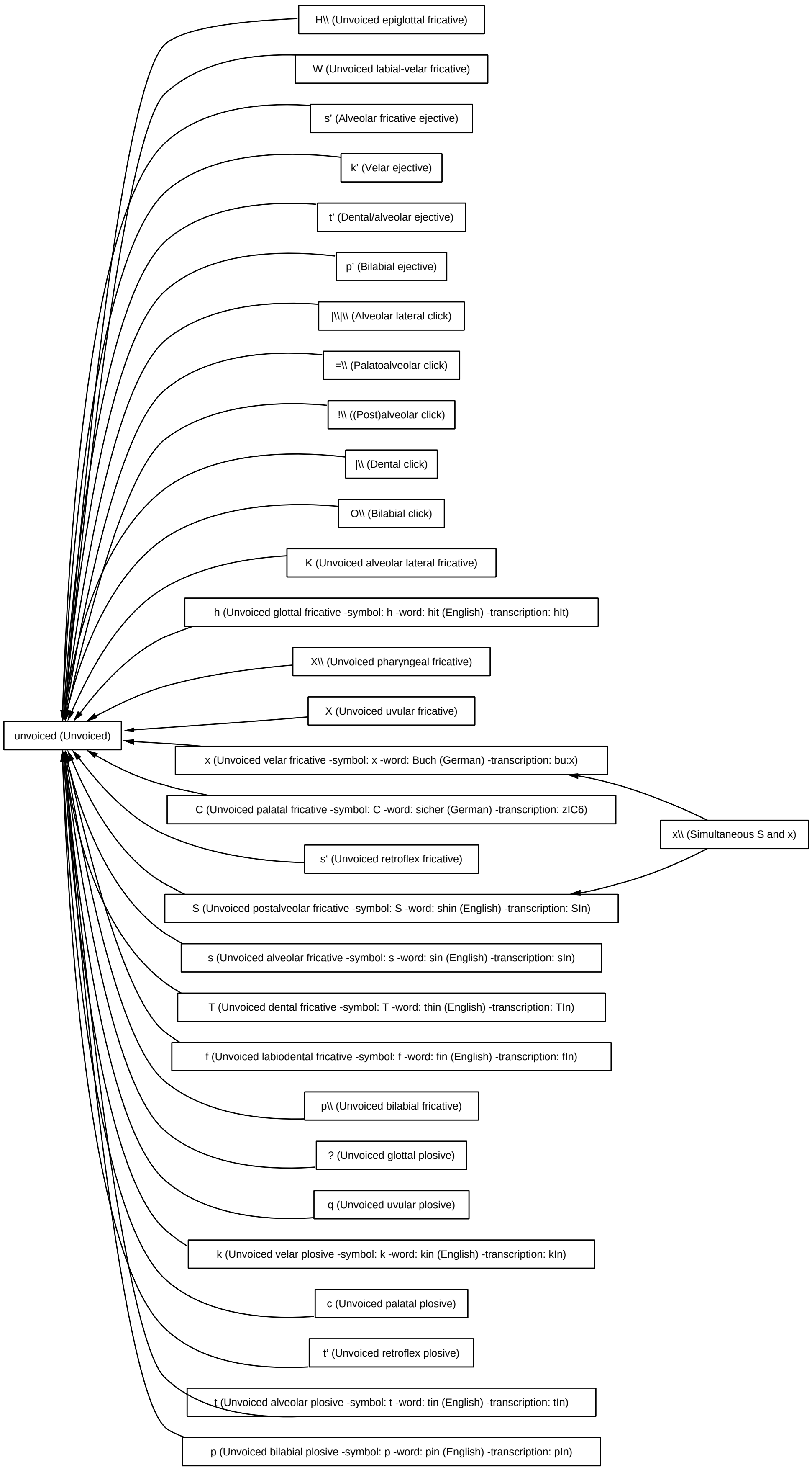


Figure G.33: Unvoiced glottis state

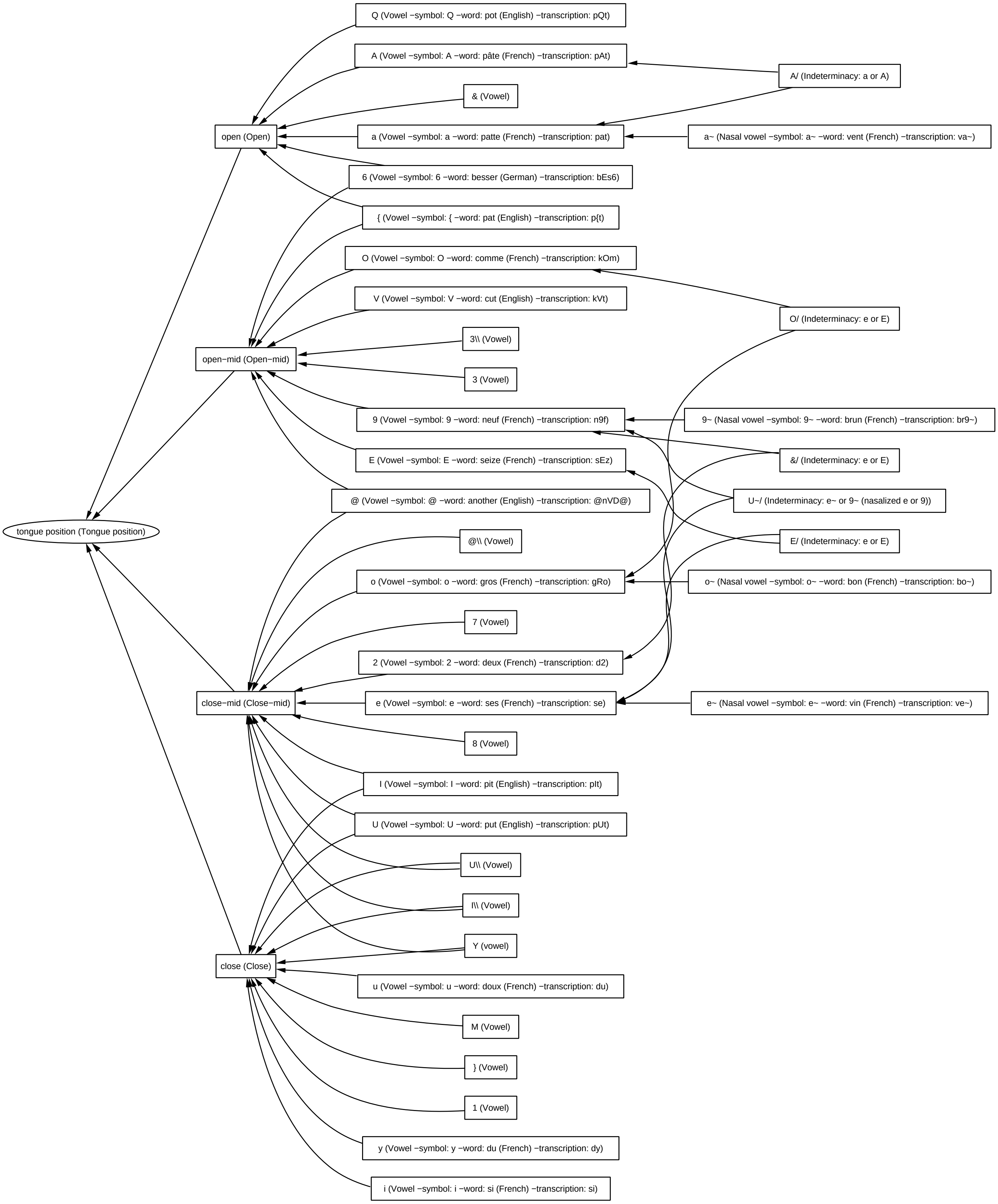


Figure G.34: Tongue position

Figure G.35: Tongue height

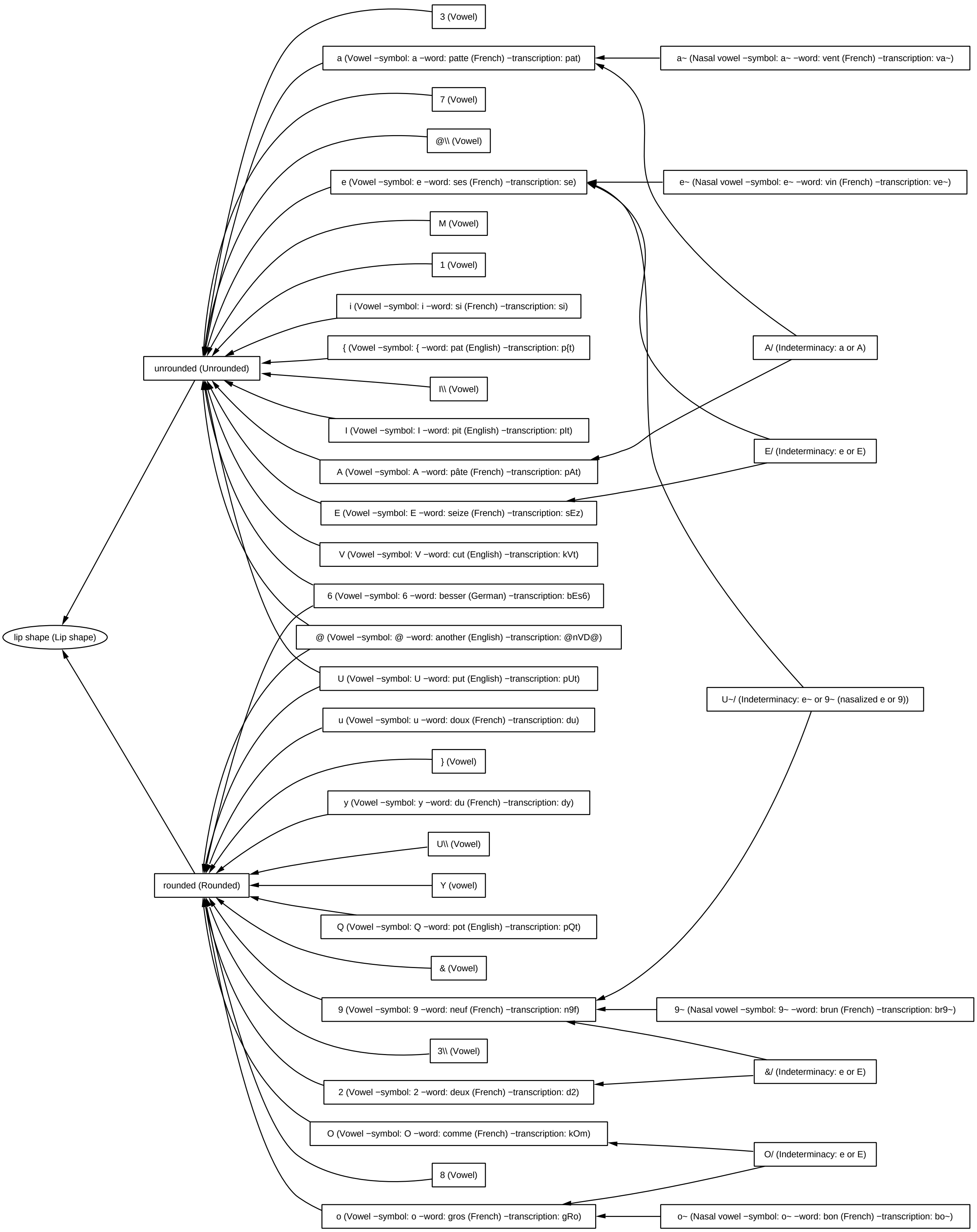


Figure G.36: Lip shape

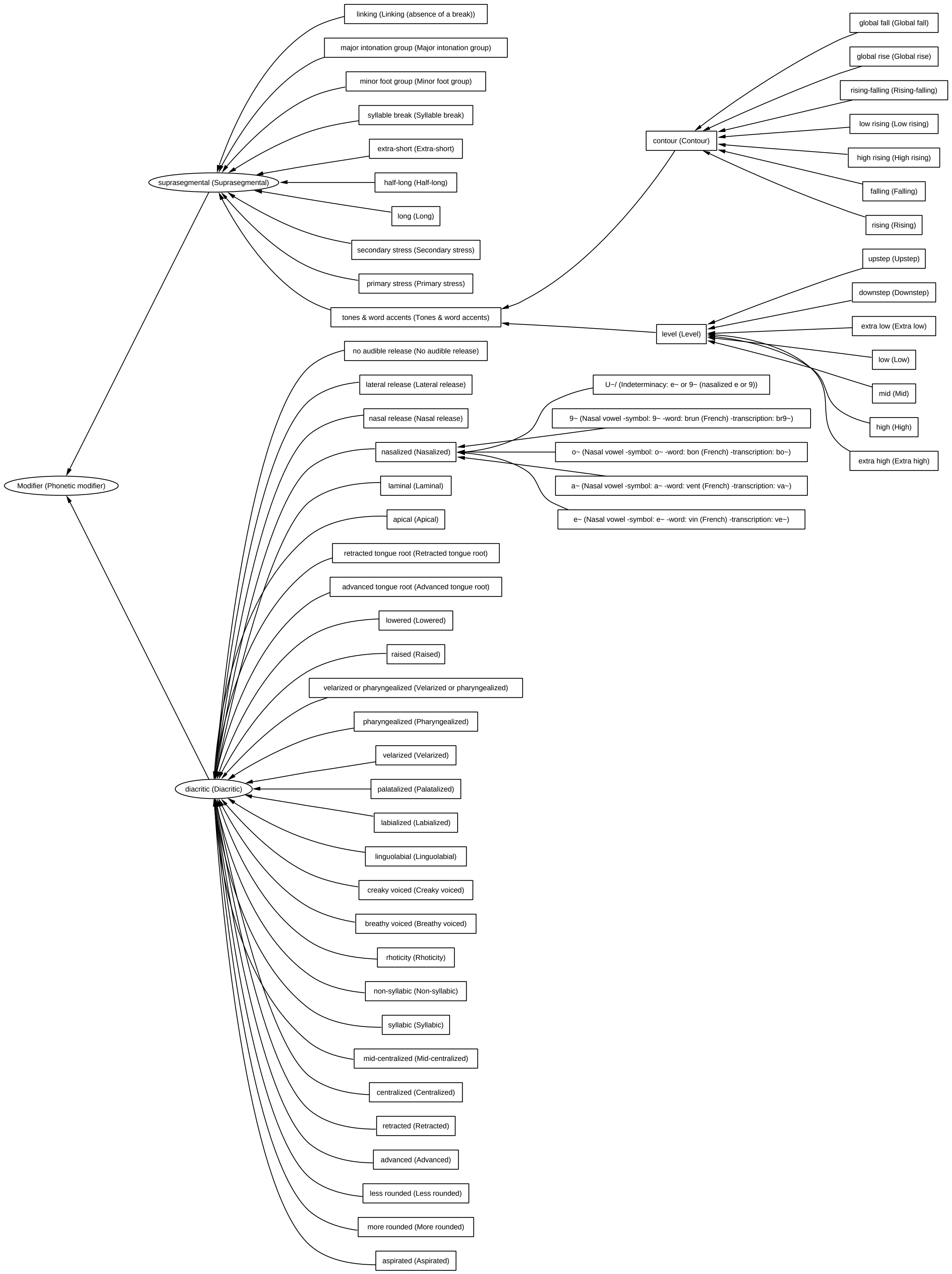


Figure G.37: Phonetic modifiers overview

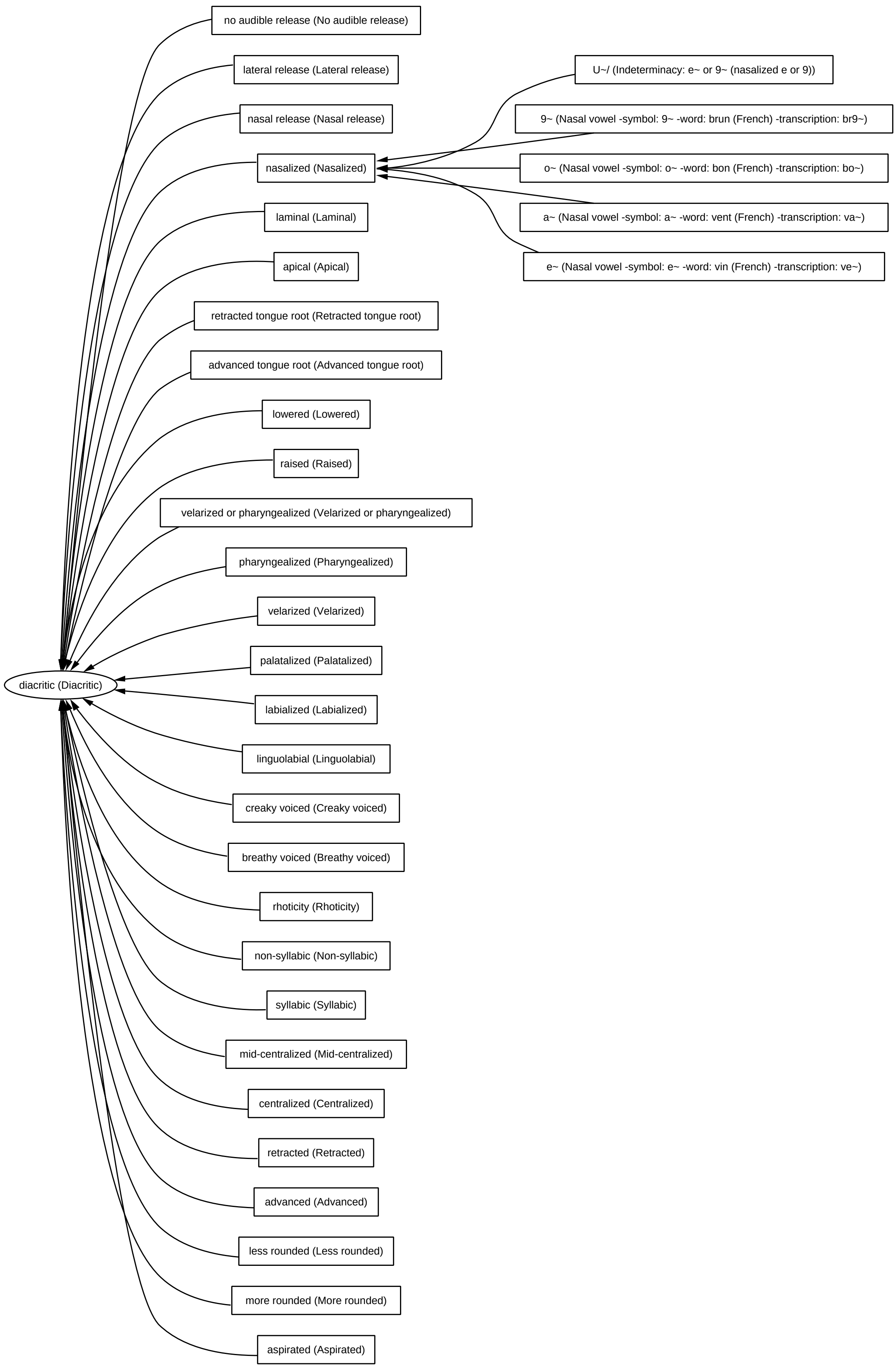


Figure G.38: Diacritic phonetic modifiers

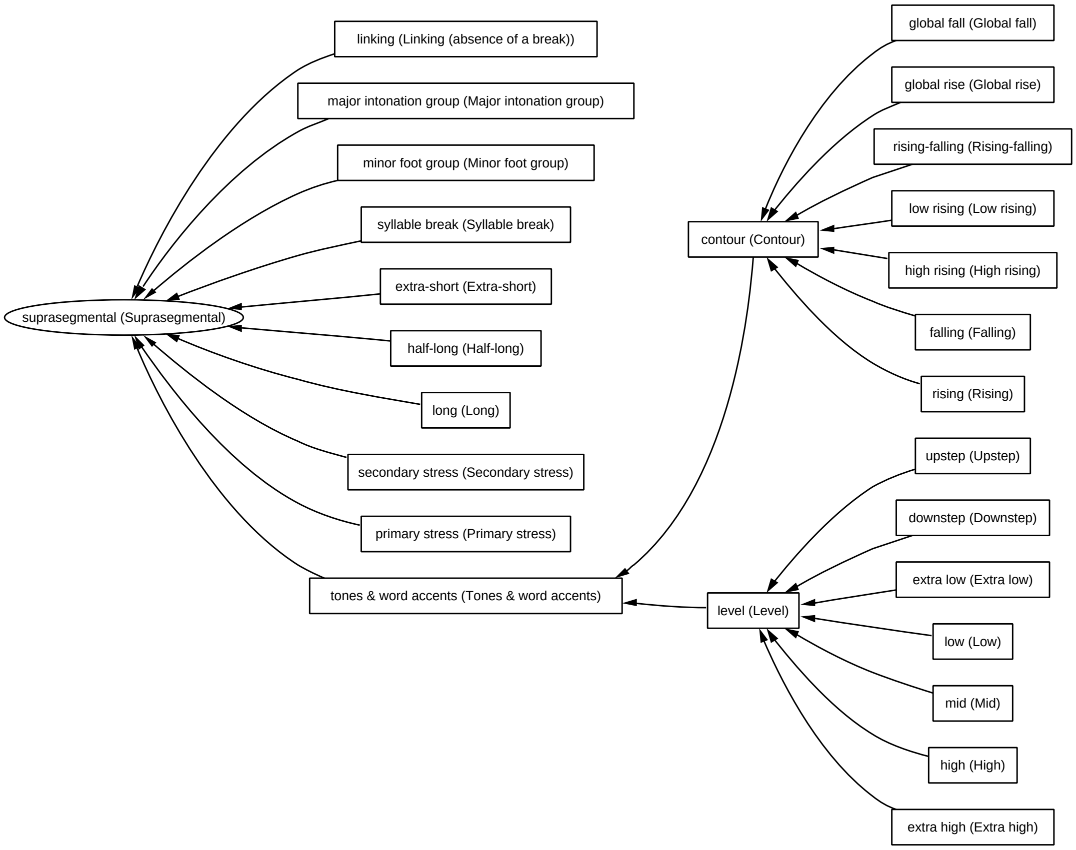


Figure G.39: Suprasegmental phonetic modifiers

References

Allen, Jonathan (1992). Overview of text-to-speech systems. In Sadaoki Furui (Ed.), *Advances in Speech Signal Processing*. New York, USA: Dekker.

ATR (2003). Real-Time Applications Team/Équipe Applications Temps-Réel, Ircam—Centre Pompidou. Web page. http://www.ircam.fr/equipes/temps-reel

Avendano, C., S. van Vuuren, and H. Hermansky (1996, October). Data based filter design for RASTA-like channel normalization in ASR. In *Proceedings of the International Conference on Spoken Language Processing (ICSLP)*, Volume 4, Philadelphia, PA, pp. 2087–2090.

Bagein, Michel, Thierry Dutoit, Nawfal Tounsi, Fabrice Malfrère, Alain Ruelle, and Dominique Wynsberghe (2001). *Le projet EULER, Vers une synthèse de parole générique et multilingue*, Volume 42. Edition Hermes Science Publication.

Baird, B., D. Blevins, and N. Zahler (1990). The Artificially Intelligent Computer Performer: The Second Generation. In *Interface – Journal of New Music Research*, Number 19, pp. 197–204.

Baird, B., D. Blevins, and N. Zahler (1993). Artificial Intelligence and Music: Implementing an Interactive Computer Performer. *Computer Music Journal 17*(2), 73–79.

Baudoin, Geneviève, François Capman, Jan Černocký, Fadi El Chami, Maurice Charbit, Gérard Chollet, and Dijana Petrovska-Delacrétaz (2002). Advances in very low bit rate speech coding using recognition and synthesis techniques. *Lecture Notes in Computer Science 2448*, 269–276.

Baudoin, Geneviève, Jan Černocký, Gérard Chollet, and Philippe Gournay (2000). Codage de la parole à très bas débit. *Annales des Télécommunications, 55*(9–10).

Beauchamp, J. W. (1975). Analysis and synthesis of cornet tones using nonlinear interharmonic relationships. *Journal of the Audio Engineering Society (AES) 23*(10), 778–795.

Beauchamp, J. W. (1980). Analysis of simultaneous mouthpiece and output waveforms of wind instruments. *Journal of the Audio Engineering Society (AES)*. Preprint No. 1626.

Beazley, David, William Fulton, Matthias Köppe, Lyle Johnson, Richard Palmer, Craig Files, Art Yerkes, and Jonah Beckford (2003, March). SWIG 1.3 Development Documentation. http://www.swig.org/Doc1.3.

Beazley, David M., David Fletcher, and Dominique Dumont (1998). Perl Extension Building with SWIG. In *Proceedings of the O'Reilly Perl Conference*.

Beeferman, Doug, Adam Berger, and John D. Lafferty (1999). Statistical models for text segmentation. *Machine Learning 34*(1–3), 177–210.

Bellini, P. and Paolo Nesi (2001, November). WEDELMUSIC Format: An XML Music Notation Format for Emerging Applications. In *First International Conference on WEB Delivering of Music (WEDELMUSIC'01)*, Florence, Italy, pp. 79. http://www.wedelmusic.org/

Benade, A. H. (1976). *Fundamentals of Musical Acoustics*. Oxford University Press.

Bender, O., K. Macherey, F. J. Och, and H. Ney (2003, April). Comparison of Alignment Templates and Maximum Entropy Models for Natural Language Understanding. In *Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, Budapest, Hungary, pp. 11–18.

Bennett, G. and X. Rodet (1989). Synthesis of the singing voice. In M. V. Mathews and J. R. Pierce (Eds.), *Current Directions in Computer Music Research*. MIT Press.

Berndtsson, G. (1995). The KTH Rule Systeme for Singing Synthesis. *Computer Music Journal 20*(1), 76–91.

Beutnagel, M., A. Conkie, J. Schroeter, Y. Stylianou, and A. Syrdal (1999, March). The AT&T Next-Gen TTS System. In *Joint Meeting of ASA, EAA, and DAGA*, Berlin, Germany. .

Beutnagel, M., M. Mohri, and M. Riley (1999, September). Rapid unit selection from a large speech corpus for concatenative speech synthesis. In *Proceedings of the European Conference on Speech Communication and Technology (EUROSPEECH)*, Budapest, Hungary.

Black, Alan and Paul Taylor (1994). CHATR: A Generic Speech Synthesis System. In *COLING94*, Kyoto, Japan.

Black, Alan and Paul Taylor (1997a, January). The Festival Speech Synthesis System: System Documentation (1.1.1). Technical Report HCRC/TR-83, Human Communication Research Centre, University of Edinburgh. .

Black, Alan, Paul Taylor, and Richard Caley (1998, December). The Festival Speech Synthesis System: System Documentation (1.3.1). Technical Report HCRC/TR-83, Human Communication Research Centre, University of Edinburgh. .

Black, Alan W. and Nick Campbell (1995, September). Optimising selection of units from speech databases for concatenative synthesis. In *Proceedings of the European Conference on Speech Communication and Technology (EUROSPEECH)*, Volume 1, Madrid, Spain, pp. 581–584. .

Black, Alan W. and Kevin A. Lenzo (2001). Optimal data selection for unit selection synthesis. In *4thESCA Workshop on Speech Synthesis*, Scotland.

Black, Alan W. and Ariadna Font Llitjós (2002). Unit selection without a phoneme set. In *IEEE TTS Workshop*, Santa Monica, CA.

Black, Alan W. and Paul Taylor (1997b, September). Automatically clustering similar units for unit selection in speech synthesis. In *Proceedings of the European Conference on Speech Communication and Technology (EUROSPEECH)*, Rhodes, Greece, pp. 601–604. .

Bloch, J. and Roger B. Dannenberg (1985). Real-Time Accompaniment of Polyphonic Keyboard Performance. In *Proceedings of the International Computer Music Conference (ICMC)*, pp. 279–290.

Blouin, Christophe (2003, December). *Sélection des unités pour la synthèse vocale par concaténation*. Ph. D. thesis, France Télécom R&D Lannion, LIMSI.

Blouin, Christophe and P.C. Bagshaw (2003, April). A method of unit pre-selection for speech synthesis based on acoustic clustering and decision trees. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Hong Kong, China.

Blouin, Christophe, O. Rosec, P.C. Bagshaw, et al. (2002, September). Concatenation cost calculation and optimisation for unit selection in TTS. In *IEEE TTS Workshop*, Santa-Monica, California, USA.

Böhm, Carola, Donald MacLellan, and Cordy Hall (2000). MuTaTeD'II: A System for Music Information Retrieval of Encoded Music. In *Proceedings of the International Computer Music Conference (ICMC)*, Berlin. ICMA.

Bonada, Jordi, Oscar Celma, Alex Loscos, Jaume Ortola, Xavier Serra, Yasuo Yoshioka, Hiraku Kayama, Yuji Hisaminato, and Hideki Kenmochi (2001, September). Singing voice synthesis combining excitation plus resonance and sinusoidal plus residual models. In *Proceedings of the International Computer Music Conference (ICMC)*, Havana, Cuba.

Booch, Grady, James Rumbaugh, and Ivar Jacobson (1999). *The Unified Modeling Language User Guide*. Addison Wesley.

Bouvier, Alain and Michel George (1983). *Dictionnaire des Mathématiques* (2nd ed.). Paris, France: Presses Universitaires de France.

Bozkurt, Baris, Thierry Dutoit, Romain Prudon, Christophe D'Alessandro, and Vincent Pagel (2002, September). Improving Quality of MBROLA Synthesis for Non-Uniform Units Synthesis. In *Proceedings of the IEEE TTS 2002 Workshop*, Santa Monica.

Breiman, L., J. Friedman, R. Olshen, and C. Stone (1984). *Classification and Regression Trees*. Monterey, CA: Wadsworth and Brooks.

Bryson, Joanna (1995). The Reactive Accompanist: Adaptation and Behavior Decomposition in a Music System. In Luc Steels (Ed.), *The Biology and Technology of Intelligent Autonomous Agents*. Springer-Verlag: Heidelberg, Germany.

Bulut, Murtaza, Shrikanth S. Narayanan, and Ann K. Syrdal (2002, September). Expressive speech synthesis using a concatenative synthesizer. In *ICSLP*, Denver, Colorado, USA.

Bußmann, Hadumod (1990). *Lexikon der Sprachwissenschaft* (2nd ed.). Stuttgart, Germany: Kröner.

Cardle, Marc, Stephens Brooks, and Peter Robinson (2003). Audio and user directed sound synthesis. In *Proceedings of the International Computer Music Conference (ICMC)*, Singapore.

Cemgil, A. T., H. Kappen, P. Desain, and H. Honing (2001). On Tempo Tracking: Tempogram Representation and Kalman Filtering. *29*(4), 259–273.

Černocký, Jan, Geneviève Baudoin, and Gérard Chollet (1998). Very low bit rate segmental speech coding using automatically derived units. In *Proceedings Radioelektronika*, Brno, Czech Republic.

Černocký, Jan, Geneviève Baudoin, Dijana Petrovska-Delacrétaz, Jean Hennebert, and Gérard Chollet (1998, September). Automatically Derived Speech Units: Applications to Very Low Rate Coding and Speaker Verification. In Petr Sojka, Václav Matoušek, Karel Pala, and Ivan Kopeček (Eds.), *Proceedings of the First Workshop on Text, Speech, Dialogue (TSD)*, Brno, Czech Republic, pp. 183–188. Masaryk University Press.

Chion, Michel (1995). *Guide des objets sonores*. Paris, France: Buchet/Chastel.

Chollet, Gérard, Jan Černocký, A. Constantinescu, S. Deligne, and F. Bimbot (1999). *Towards ALISP: a Proposal for Automatic Language Independent Speech Processing*, pp. 375–388. NATO ASI. Springer Verlag.

Clark, John E. and Colin Yallop (1996). *An Introduction to Phonetics and Phonology*. Oxford: Blackwell.

Clarke, Michael and Xavier Rodet (2003). Real-time fof and fog synthesis in msp and its integration with psola. In *Proceedings of the International Computer Music Conference (ICMC)*, Singapore.

Clarke, Michael J. (1996). TIM(br)E: Compositional Approaches to FOG Synthesis. In *Proceedings of the International Computer Music Conference (ICMC)*, Hong Kong, pp. 375–377.

Clarke, Michael J. (2000). FOF and FOG Synthesis in Csound. In R. Boulanger (Ed.), *The Csound Book*, pp. 293–306. MIT Press.

Codd, E. F. (1970, June). A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM 13*(6).

Codognet, Philippe and Daniel Diaz (2001, November). Yet another local search method for constraint solving. In *AAAI Symposium*, North Falmouth, Massachusetts.

Cook, Perry R. (1989). Synthesis of the Singing Voice Using a Physically Parameterized Model of the Human Vocal Tract. In *Proceedings of the International Computer Music Conference (ICMC)*, Columbus, Ohio, USA, pp. 69–72.

Cook, Perry R. (1991). *Identification of Control Parameters in an Articulatory Vocal Tract Model with Applications to the Synthesis of Singing*. Ph. D. thesis, CCRMA, Stanford University.

Cook, Perry R. (1996). Singing Voice Synthesis: History, Current Work, and Future Directions. *Computer Music Journal 20*(2).

Cook, Perry R. (Ed.) (1999). *Music, Cognition and Computerized Sound: an Introduction to Psychoacoustics*. Cambridge, Massachusetts, USA: MIT Press.

Cope, David (1996). *Experiments in Musical Intelligence*. Madison, WI, USA: A-R Editions.

Cope, David (2003). Experiments in Musical Intelligence. Web page. `http://arts.ucsc.edu/faculty/cope/emmy.html`.

Cover, Robin (2000). The XML Cover Pages. Web page. `http://www.oasis-open.org/cover/xml.html`.

Cronk, Andrew E. and Michael W. Macon (1998, November). Optimized Stopping Criteria for Tree-Based Unit Selection in Concatenative Synthesis. In *Proceedings of the International Conference on Spoken Language Processing (ICSLP)*, Volume 5, pp. 1951–1955. .

Dannenberg, Roger, Jonathan Foote, George Tzanetakis, and Christopher Weare (2001, September). Panel: New directions in music information retrieval. In *Proceedings of the International Computer Music Conference (ICMC)*, Havana, Cuba.

Dannenberg, R. and N. Hu (2003). Polyphonic audio matching for score following and intelligent audio editors. In *Proceedings of the International Computer Music Conference (ICMC)*, Singapore.

Dannenberg, Roger and Patrick van de Lageweg (2001, September). A system supporting flexible distributed real-time music processing. In *Proceedings of the International Computer Music Conference (ICMC)*, Havana, Cuba.

Dannenberg, Roger B. (1984). An On-Line Algorithm for Real-Time Accompaniment. In *Proceedings of the International Computer Music Conference (ICMC)*, pp. 193–198.

Dannenberg, Roger B. and Istvan Derenyi (1998, September). Combining Instrument and Performance Models for High-Quality Music Synthesis. *Journal of New Music Research 27*(3), 211–238.

Dannenberg, Roger B. and B. Mont-Reynaud (1987). Following an Improvisation in Real Time. In *Proceedings of the International Computer Music Conference (ICMC)*, pp. 241–248.

Dannenberg, Roger B. and Mukaino (1988). New Techniques for Enhanced Quality of Computer Accompaniment. In *Proceedings of the International Computer Music Conference (ICMC)*, pp. 243–249.

Date, C. J. (1981). *An Introduction to Database Systems* (3rd ed.), Volume I of *The Systems Programming Series*. Addison–Wesley.

Date, C. J. and Hugh Darwen (1997). *A Guide to the SQL Standard: A user's guide to the standard database language SQL* (4th ed.). Addison–Wesley.

Déchelle, François, Maurizio De Cecco, Enzo Maggi, and Norbert Schnell (1999, October). *jMax* Recent Developments. In *Proceedings of the International Computer Music Conference (ICMC)*, Beijing, China.

de Cheveigné, Alain (2002). Two Voice Fundamental Frequency Estimation. *Journal of the Acoustical Society of America (JASA) 111*.

de Cheveigné, Alain and Nathalie Henrich (2002). Fundamental Frequency Estimation of Musical Sounds. *Journal of the Acoustical Society of America (JASA)*.

de Cheveigné, Alain and Hideki Kawahara (2002). YIN, a Fundamental Frequency Estimator for Speech and Music. *Journal of the Acoustical Society of America (JASA) 111*, 1917–1930.

Déchelle, François, Norbert Schnell, Ricardo Borghesi, and Nicolas Orio (2000, August). The *jMax* Environment: An Overview of New Features. In *Proceedings of the International Computer Music Conference (ICMC)*, Berlin, Germany.

Depalle, Philippe, Guillermo García, and Xavier Rodet (1994). A Virtual Castrato (!?). In *Proceedings of the ICMC*. .

D'haes, Wim and Xavier Rodet (2001, September). Automatic estimation of control parameters: An instance-based learning approach. In *Proceedings of the International Computer Music Conference (ICMC)*, Havana, Cuba.

D'haes, Wim and Xavier Rodet (2002, September). Physical constraints for the control of a physical model of a trumpet. In *Proceedings of the COST-G6 Conference on Digital Audio Effects (DAFx)*, Hamburg, Germany, pp. 157–162.

D'haes, Wim and Xavier Rodet (2003, September). A new estimation technique for determining the control parameters of a physical model of a trumpet. In *Proceedings of the COST-G6 Conference on Digital Audio Effects (DAFx)*, London, UK.

Dixon, Simon (2001). An empirical comparison of tempo trackers. In *8^{th}Brazilian Symposium on Computer Music*, Fortaleza, Brazil.

Dogil, Grzegorz (1995). Phonetic correlates of word stress. *AIMS Phonetik (Working Papers of the Department of Natural Language Processing) 2*(2). .

Dorran, David and Robert Lawlor (2003, September). An efficient audio time-scale modification algorithm for use in a subband implementation. In *Proceedings of the COST-G6 Conference on Digital Audio Effects (DAFx)*, London, UK.

Doval, Boris (1994, March). *Estimation de la fréquence fondamentale des signaux sonores*. Ph. D. thesis, LAFORIA, Université Paris VI.

DuCharme, Bob (1999). *XML: The Annotated Specification*. The Charles F. Goldfarb series on open information management. Upper Saddle River, NJ 07458, USA: Prentice-Hall PTR.

Dufour, Denis, Jean-Christophe Thomas, et al. (1999). *Ouir, entendre, écouter, comprendre après Schaeffer*. Paris, France: Buschet/Chastel/INA-GRM.

Durbin, R., S. Eddy, A. Krogh, and G. Mitchison (1998). *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press.

Dutoit, Thierry (1993, October). *High Quality Text-To-Speech Synthesis of the French Language*. Ph. D. thesis, Faculté Polytechnique de Mons, TCTS Lab, Mons, Belgium.

Dutoit, Thierry (1994, April). High quality text-to-speech synthesis: a comparison of four candidate algorithms. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Adelaide, Austrailia, pp. I–565–I–568.

Dutoit, Thierry (1999, August). The MBROLIGN Project: Towards a Large Repository of Aligned Text-to-Speech. Web page. `http://tcts.fpms.ac.be/synthesis/mbrolign/mbrolign.html`.

Dutoit, Thierry (2000, November). The EULER Project. Web page. `http://tcts.fpms.ac.be/synthesis/euler`.

Dutoit, Thierry (2003, October). The MBROLA Project: Towards a Freely Available Multilingual Speech Synthesizer. Web page. `http://tcts.fpms.ac.be/synthesis`.

Dutoit, T., F. Malfrère, V. Pagel, M. Bagein P. Mertens, A. Ruelle, and A. Gilman (1998, September). EULER: Multi-Lingual Text-to-Speech Project. In Petr Sojka, Václav Matoušek, Karel Pala, and Ivan Kopeček (Eds.), *Proceedings of the First Workshop on Text, Speech, Dialogue — TSD'98*, Brno, Czech Republic, pp. 27–32. Masaryk University Press.

Dutoit, T., V. Pagel, N. Pierret, F. Bataille, and O. V. der Vrecken (1996, October). The MBROLA project: Towards a set of high quality speech synthesizers free of use for non commercial purposes. In *Proceedings of the International Conference on Spoken Language Processing (ICSLP)*, Volume 3, Philadelphia, PA, pp. 1393–1396.

Duxbury, Chris, Juan Pablo Bello, Mike Davies, and Mark Sandler (2003, September). Complex domain onset detection for musical signals. In *Proceedings of the COST-G6 Conference on Digital Audio Effects (DAFx)*, London, UK.

Fitz, Kelly, Lippold Haken, and Paul Christensen (2000a). A New Algorithm for Bandwidth Association in Bandwidth-Enhanced Additive Sound Modeling. In *Proc. ICMC*, Berlin.

Fitz, Kelly, Lippold Haken, and Paul Christensen (2000b). Transient Preservation under Transformation in an Additive Sound Model. In *Proceedings of the International Computer Music Conference (ICMC)*, Berlin.

Fletcher, N. H. and A. Tarnopolsky (1999). Blowing pressure power and spectrum in trumpet playing. *J. Acoust. Soc. Am. 105*(2).

Foote, Jonathan (1999, January). An overview of audio information retrieval. *Multimedia Systems 7*(1), 2–10.

Forney (1973, March). The viterbi algorithm. *Proceedings of the IEEE 61*, 268–278.

Fukada, T., K. Tokuda, T. Kobayashi, and S. Imai (1992). An adaptive algorithm for mel-cepstral analysis of speech. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 137–140.

Gershenfeld, N.A., B. Schoner, and E. Métois (1999). Cluster-weighted modeling for time series analysis. *Nature* (37), 329–332.

Ghitza, O. and M. M. Sondhi (1997). On the perceptual distance between two speech segments. In *Journal of the Acoustical Society of America*, Volume 101, pp. 522–529.

Gómez, E., F. Gouyon, P. Herrera, and X. Amatriain (2003a). Mpeg-7 for content-based music processing. In *Proceedings of 4th WIAMIS-Special session on Audio Segmentation and Digital Music*.

Gómez, E., F. Gouyon, P. Herrera, and X. Amatriain (2003b). Using and enhancing the current mpeg-7 standard for a music content processing tool. In *Proceedings of Audio Engineering Society, 114th Convention*.

Gouyon, Fabien (2000). Extraction de descripteurs rythmiques dans des extraits de musiques populaires polyphoniques. Master's thesis, Ircam – Centre Georges Pompidou, Paris, France. Rapport de stage DEA ATIAM.

Gouyon, F. and P. Herrera (2003). A beat induction method for musical audio signals. In *Proceedings of 4^{th} WIAMIS-Special session on Audio Segmentation and Digital Music*, London, UK.

Gray, J. and A. Reuter (1993). *Transaction Processing: Concepts and Techniques*. San Mateo (CA), USA: Morgan Kaufmann Publishers.

Gribonval, R., L. Benaroya, E. Vincent, and C. Févotte (2003). Proposals for performance measurement in source separation. In *Proc. 4^{th} Symposium on Independent Component Analysis and Blind Source Separation (ICA 2003)*, Nara, Japan.

Grubb, Lorin and Roger B. Dannenberg (1994). Automating Ensemble Performance. In *Proceedings of the International Computer Music Conference (ICMC)*, pp. 63–69.

Grubb, Lorin and Roger B. Dannenberg (1997). A Stochastic Method of Tracking a Vocal Performer. In *Proceedings of the International Computer Music Conference (ICMC)*, pp. 301–308.

Grubb, Lorin and Roger B. Dannenberg (1998). Enhanced Vocal Performance Tracking Using Multiple Information Sources. In *Proceedings of the International Computer Music Conference (ICMC)*, pp. 37–44.

GUIDO (2003). The GUIDO Music Notation Format Homepage. Web page. `http://www.salieri.org/guido`.

Hainsworth, S. and M. Macleod (2003). Onset detection in musical audio signals. In *Proceedings of the International Computer Music Conference (ICMC)*, Singapore.

Hansen, J. H. L. and D. T. Chappell (1998, September). An auditory-based distortion measure with application to concatenative speech synthesis. In *IEEE Trans. on Speech and Audio Processing*, Volume 6, pp. 489–495.

Hazel, Steven (2001). Soundmosaic. web page. `http://thalassocracy.org/soundmosaic`.

Helen, Marko and Tuomas Virtanen (2003, September). Perceptually motivated parametric representation for harmonic sounds for data compression purposes. In *Proceedings of the COST-G6 Conference on Digital Audio Effects (DAFx)*, London, UK.

Henrich, Nathalie (2001, November). *Etude de la source glottique en voix parlée et chantée*. Ph. D. thesis, Université Paris 6, Paris, France.

Hermansky, Hynek (1998, September). Data-Driven Speech Analysis for ASR. In Petr Sojka, Václav Matoušek, Karel Pala, and Ivan Kopeček (Eds.), *Proceedings of the First Workshop on Text, Speech, Dialogue — TSD'98*, Brno, Czech Republic, pp. 213–218. Masaryk University Press. .

Hermansky, Hynek (1999, September 13–17). Data-driven analysis of speech. In Václav Matoušek, Pavel Mautner, Jana Ocelíková, and Petr Sojka (Eds.), *Proceedings of the 2nd International Workshop on Text, Speech and Dialogue (TSD-99)*, Volume 1692 of *Lecture Notes on Artificial Intelligence (LNAI)*, Berlin, pp. 10–18. Springer.

Hermansky, H., B. A. Hanson, and H. Wakita (1985). Perceptually based linear predictive analysis of speech. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 509–512.

Hess, Wolfgang (1996, December). Maschinelle sprachsynthese. *Spektrum der Wissenschaft*.

Hélie, Thomas (2002, December). *Modélisation physique d'instruments de musique en systèmes dynamiques et inversion*. Ph. D. thesis, Université Paris XI.

Holmes, J. N. (1983). Formant synthesizers: Cascade or Parallel. In *Speech Communication*, Volume 2, pp. 251–273.

Holmes, J. N. (1995). *Speech Synthesis and Recognition*. London, UK: Chapman & Hall.

Hoos, H. H., K. A. Hamel, K. Renz, and J. Kilian (1998). The GUIDO Music Notation Format — A Novel Approach for Adequately Representing Score-level Music. In *Proceedings of the International Computer Music Conference (ICMC)*, Ann Arbor, MI, USA, pp. 451–454. ICMA.

Hoos, Holger H., Kai Renz, and Marko Görg (2001, October). Guido/mir — an experimental musical information retrieval system based on guido music notation. In *2^{nd} Annual International Symposium on Music Information Retrieval (ISMIR)*, Bloomington, Indiana, USA.

Hoskinson, Reynald and Dinesh Pai (2001, September). Manipulation and resynthesis with natural grains. In *Proceedings of the International Computer Music Conference (ICMC)*, Havana, Cuba.

Huang, Fu Jie, Eric Cosatto, and Hans Peter Graf (2002, May). Triphone based unit selection for concatenative visual speech synthesis.

Hui-Ling, L. (2002). *Toward a High Quality Singing Synthesizer with Vocal Texture Control*. Ph. D. thesis, Stanford University.

Hunt, Andrew J. and Alan W. Black (1996, May). Unit selection in a concatenative speech synthesis system using a large speech database. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Atlanta, GA, pp. 373–376. .

Hunter, Jane (1999). MPEG7 Behind the Scenes. *D-Lib Magazine 5*(9).

IPA (2003). The international phonetic association. Web page. `http://www2.arts.gla.ac.uk/IPA/ipa.html`.

ISI News and Events (2003, July). Computer Language Translation System Romances the Rosetta Stone. *ISI News and Events*. `http://www.usc.edu/isinews/stories/102.html`

ISO (1995, July). Music Description Language (SMDL). International Organization for Standardization (ISO), `http://www.oasis-open.org/cover/smdlover.html`.

Izmirli, O., R. Seward, and N. Zahler (2003). Melodic pattern anchoring for score following using score analysis. In *Proceedings of the International Computer Music Conference (ICMC)*, Singapore.

Jaillet, Florent (2000). Détection et modélisation des transitoires d'attaque rapides. Master's thesis, Ircam – Centre Georges Pompidou, Paris, France. Rapport de stage DEA ATIAM.

Jehan, Tristan (1997). Musical signal parameter estimation. Master's thesis, IFSIC, Université de Rennes, France, and Center for New Music and Audio Technologies (CNMAT), University of California, Berkeley, USA.

Jehan, Tristan and Bernd Schoner (2001a). An Audio-Driven, Spectral Analysis-Based, Perceptual Synthesis Engine. In *Proceedings of the 110^{th} Convention of the Audio Engineering Society (AES)*, Amsterdam, The Netherlands.

Jehan, Tristan and Bernd Schoner (2001b, September). An audio-driven perceptually-meaningful timbre synthesizer. In *Proceedings of the International Computer Music Conference (ICMC)*, Havana, Cuba.

Jensen, Kristoffer (1999). *Timbre Models of Musical Sounds*. Ph. D. thesis, Department of Computer Science, University of Copenhagen, Denmark.

Knuth, Donald. E. (1990). *The TeX-Book (revised)*. Addison Wesley.

Kobayashi, R. (2003). Sound clustering synthesis using spectral data. In *Proceedings of the International Computer Music Conference (ICMC)*, Singapore.

Kopeček, Jan Černocký Ivan, Geneviève Baudoin, and Gérard Chollet (1999, September 13–17). Very low bit rate speech coding: Comparison of data-driven units with syllable segments. In Václav Matoušek, Pavel Mautner, Jana Ocelíková, and Petr Sojka (Eds.), *Proceedings of the 2^{nd}International Workshop on Text, Speech and Dialogue (TSD)*, Volume 1692 of *LNAI*, Berlin, pp. 262–267. Springer.

Koutsofios, Eleftherios and Stephen C. North (1996, November). Drawing graphs with *dot*. Technical report, AT&T Bell Laboratories, Murray Hill, NJ, USA. http://www.research.att.com/sw/tools/graphviz.

Küpfmüller, K. and O. Warns (1954). Sprachsynthese aus lauten. *Nachrichtentechnische Fachberichte* (3), 28–31.

Lambert, Jean-Philippe (1999, June). A Beat Tracker for the Score Recognition. Technical report, Ircam.

Lambert, Jean-Philippe (2001a, November). Développement du programme de calcul des descripteurs de haut niveau. Technical report, Ircam – Centre Pompidou, France Télécom R&D, Paris, France. Version 1.3.

Lambert, Jean-Philippe (2001b, December). Synthèse et traitement à partir de descripteurs de son. Technical report, Ircam – Centre Pompidou, Paris, France. Version 0.2.

Lazier, Ari and Perry Cook (2003, September). MOSIEVIUS: Feature driven interactive audio mosaicing. In *Proceedings of the COST-G6 Conference on Digital Audio Effects (DAFx)*, London, UK.

Le Meur, P.Y. (1996). *Synthèse de parole par unités de taille variable*. Ph. D. thesis, Ecole Nationale Supérieure des Télécommunications (ENST).

Levine, Scott N. (1998, December). *Audio Representations for Data Compression and Compressed Domain Processing*. Ph.d. dissertation, Department of Electrical Engineering, CCRMA, Stanford University. http://www-ccrma.stanford.edu/~scottl/thesis.html.

Livshin, Arie, Geoffroy Peeters, and Xavier Rodet (2003). Studies and improvements in automatic classification of musical sound samples. In *Proceedings of the International Computer Music Conference (ICMC)*, Singapore.

Lomax, Ken (1996, May). The development of a singing synthesiser. In *3èmes Journees d'Informatique Musicale (JIM)*, Ile de Tatihou, Lower Normandy, France.

Lopez, Daniel, Francesc Martì, and Eduard Resina (1998). Vocem: An Application for Real-Time Granular Synthesis.

Loscos, A., P. Cano, and J. Bonada (1999a). Low-Delay Singing Voice Alignment to Text. In *Proceedings of the International Computer Music Conference (ICMC)*.

Loscos, A., P. Cano, and J. Bonada (1999b). Score-Performance Matching using HMMs. In *Proceedings of the International Computer Music Conference (ICMC)*, pp. 441–444.

Macon, Mike (2000, May). Flinger System Documentation. Web page. CSLU, Oregon Graduate Institute (OGI). http://www.cslu.ogi.edu/tts/flinger/fldoc.html.

Macon, Michael, Leslie Jensen-Link, James Oliverio, Mark A. Clements, and E. Bryan George (1997a). A singing voice synthesis system based on sinusoidal modeling. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 435–438. .

Macon, Michael W. (1996, October). *Speech Synthesis Based on Sinusoidal Modeling*. Ph. D. thesis, Georgia Institute of Technology.

Macon, M. W. and M. A. Clements (1995, May). Speech synthesis based on an overlap-add sinusoidal model. In *Journal of the Acoustical Society of America (JASA)*, Volume 97, pp. 3246. Pt. 2. .

Macon, Michael W. and Mark A. Clements (1996). Speech Concatenation and Synthesis Using an Overlap–Add Sinusoidal Model. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Volume 1, Atlanta, USA, pp. 361–364. .

Macon, M. W., A. E. Cronk, and J. Wouters (1998, November). Generalization and discrimination in tree-structured unit selection. In *Proceedings of the 3rd ESCA/COCOSDA International Speech Synthesis Workshop*. .

Macon, M. W., A. E. Cronk, J. Wouters, and A. Kain (1997, September). Ogireslpc: Diphone synthesizer using residual-excited linear prediction. In *Tech. Rep. CSE-97-007*. Portland, OR: Department of Computer Science, Oregon Graduate Institute of Science and Technology. .

Macon, M. W., L. Jensen-Link, J. Oliverio, M. Clements, and E. B. George (1997b). Concatenation-Based MIDI-to-Singing Voice Synthesis. In *103rdMeeting of the Audio Engineering Society*. New York. .

Maidín, Donncha Ó (1999). Common practice notation view: a score representation for the construction of algorithms. In *Proceedings of the International Computer Music Conference (ICMC)*, Beijing, China, pp. 248–251.

Malfrere, Fabrice and Thierry Dutoit (1997a, September). High quality speech synthesis for phonetic speech segmentation. In *Proceedings of the European Conference on Speech Communication and Technology (EUROSPEECH)*, Rhodes, Greece, pp. 2631–2634.

Malfrere, F. and T. Dutoit (1997b). Speech synthesis for text-to-speech alignment and prosodic feature extraction. In *Proc. ISCAS 97*, Hong-Kong, pp. 2637–2640. .

Markel, J.D. and A.H. Gray (1980). *Linear Prediction of Speech*. Springer.

Martin, Robert C. (1997). UML Tutorial Part 1: Class Diagrams. *C++ Report*. `http://www.objectmentor.com/resources/listArticles?key=topic&topic=UML`.

Mazzoni, Dominic and Roger Dannenberg (2001, September). A fast data structure for disk-based audio editing. In *Proceedings of the International Computer Music Conference (ICMC)*, Havana, Cuba.

Meron, Yoram (1999). *High Quality Singing Synthesis Using the Selection-based Synthesis Scheme*. Ph. D. thesis, University of Tokyo.

MIDI Manufacturers Association (2001, November). *The Complete MIDI 1.0 Detailed Specification* (2nd ed.). MIDI Manufacturers Association. Version 96.1.

MMA (2003). Midi manufacturers association. Web page. `http://www.midi.org/about-mma/abtmma.shtml`.

Moore, Brain C.J., Brian R. Glasberg, and Thomas Baer (1997, April). A Model for the Prediction of Thresholds, Loudness, and Partial Loudness. *54*(4).

Moore, B. C. J. (1989). *An Introduction to the Psychology of Hearing* (3rd ed.). Academic Press Limited.

Mouillet, Vincent (2001). Prise en compte des informations temporelles dans les modèles de markov cachés. Rapport de stage, Ircam.

MPEG (2003). MPEG-7 "Multimedia Content Description Interface" Documentation. Web page. `http://www.darmstadt.gmd.de/mobile/MPEG7`

Mullon, Pascal, Yann Geslin, and Max Jacob (2002, September). Ecrins: an audio-content description environment for sound samples. In *Proceedings of the International Computer Music Conference (ICMC)*, Göteborg, Sweden.

Nakajima, S. (1994, September). Automatic synthesis unit generation for English speech synthesis based on multi-layered context oriented clustering. In *Speech Communication*, Volume 14, pp. 313.

Newcomb, S. R. (1990). Explanatory cover material for section 7.2 of X3V1.8M/SD-7, fifth draft (music description language). In J. Moline, D. Benigni, and J. Baronas (Eds.), *Proceedings of the Hypertext Standardization Workshop (NIST SP 500-178)*, Gaithersburg, MD, USA, pp. 179–188. National Institute for Standards and Technology.

Newcomb, Steven R. (1991, July). Standards: Standard Music Description Language complies with hypermedia standard. *Computer 24*(7), 76–79.

NIFF Consortium (1995, July). NIFF 6a: Notation Interchange File Format. Technical report, NIFF Consortium.

Och, Franz Josef and Hermann Ney (2002, July). Discriminative Training and Maximum Entropy Models for Statistical Machine Translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, Philadelphia, PA, pp. 295–302. Best paper award.

Oppenheim, Alan V. (Ed.) (1978). *Applications of Digital Signal Processing*, Chapter Digital Processing of Speech, pp. 117–168. Prentice–Hall.

Oppenheim, Alan V. and Ronald W. Schafer (1975). *Digital Signal Processing*. Prentice–Hall.

Orio, Nicola and François Déchelle (2001). Score Following Using Spectral Analysis and Hidden Markov Models. In *Proceedings of the International Computer Music Conference (ICMC)*, Havana, Cuba. International Computer Music Association.

Orio, Nicola, Serge Lemouton, Diemo Schwarz, and Norbert Schnell (2003, May). Score Following: State of the Art and New Developments. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*, Montreal, Canada.

Orio, Nicola and Diemo Schwarz (2001). Alignment of Monophonic and Polyphonic Music to a Score. In *Proceedings of the International Computer Music Conference (ICMC)*, Havana, Cuba.

Oswald, John (1993). Plexure. CD. `http://www.interlog.com/ vacuvox/xdiscography.html#plexure`.

Oswald, John (1999). Plunderphonics. web page. `http://www.plunderphonics.com`.

Pachet, François, Pierre Roy, and Daniel Cazaly (2000). A combinatorial approach to content-based music selection. *IEEE MultiMedia 7*(1), 44–51.

Peeters, Geoffroy (1998, May). Analyse-Synthèse des sons musicaux par la méthode PSOLA. In *Journées en Informatique Musicale*, Agelonde (France). .

Peeters, Geoffroy (2001, July). *Modèles et modélisation du signal sonore adaptés à ses caractéristiques locales*. Ph. D. thesis, Université Paris VI.

Peeters, Geoffroy, Steven McAdams, and Perfecto Herrera (2000, August). Instrument Description in the Context of MPEG-7. In *Proceedings of the International Computer Music Conference (ICMC)*, Berlin. .

Peeters, Geoffroy and Xavier Rodet (1999a, November). Non-Stationary Analysis/Synthesis using Spectrum Peak Shape Distortion, Phase and Reassigned Spectrum. In *International Conference on Signal Processing Applications & Technology (ICSPAT)*, Orlando. .

Peeters, Geoffroy and Xavier Rodet (1999b, October). SINOLA: A New Analysis/Synthesis Method using Spectrum Peak Shape Distortion, Phase and Reassigned Spectrum. In *Proceedings of the International Computer Music Conference (ICMC)*, Beijing. .

Peeters, Geoffroy and Xavier Rodet (2003a, September). Hierarchical gaussian tree with inertia ratio maximization for the classification of large musical instrument databases. In *Proceedings of the COST-G6 Conference on Digital Audio Effects (DAFx)*, London, UK.

Peeters, Geoffroy and Xavier Rodet (2003b). Signal-based music structure discovery for music audio summary generation. In *Proceedings of the International Computer Music Conference (ICMC)*, Singapore.

Peterson, G. E. and E. Sievertsen (1960). Objectives and techniques in speech synthesis. *Language and Speech* (3), 84–95.

Pfister, B. and C. Traber (1994). Text-to-speech synthesis: An introduction and a case study. In Eric Keller (Ed.), *Fundamentals of Speech Synthesis and Speech Recognition: Basic Concepts, State of the Art and Future Challenges*. Chichester: Wiley.

Pielemeier, W. J. and G. H. Wakefield (1996). A High Resolution Time–Frequency Representation for Musical Instrument Signals. *Journal of the Acoustical Society of America (JASA) 99*(4), 2382–2396.

Pols, Louis C. W. (1992). Quality assessment of text-to-speech synthesis by rule. In Sadaoki Furui (Ed.), *Advances in Speech Signal Processing.* New York, USA: Dekker.

Portele, Thomas, Florian Hoefer, and Wolfgang J. Hess (1996). A mixed inventory structure for german concatenative synthesis. In Jan P. H. van Santen, Richard W. Sproat, Joseph P. Olive, and Julia Hirschberg (Eds.), *Progress in Speech Synthesis*, pp. 263–277. New York, USA: Springer-Verlag.

Prudham, Boris (2002). Estimation de la fréquence fondamentale d'un signal. Technical report, Université de Besançon.

Prudon, Romain (2003). *A selection/concatenation TTS synthesis system.* Ph. D. thesis, LIMSI, Université Paris XI, Orsay, France.

Prudon, Romain and Christophe d'Alessandro (2001). A selection/concatenation TTS synthesis system: Databases developement, system design, comparative evaluation. In *4^{th} Speech Synthesis Workshop*, Pitlochry, Scotland.

Puckette, Miller (1990). EXPLODE: A User Interface for Sequencing and Score Following. In *Proceedings of the International Computer Music Conference (ICMC)*, pp. 259–261.

Puckette, Miller (1995). Score Following Using the Sung Voice. In *Proceedings of the International Computer Music Conference (ICMC)*, pp. 199–200.

Puckette, Miller and Cort Lippe (1992). Score Following in Practice. In *Proceedings of the International Computer Music Conference (ICMC)*, pp. 182–185.

Rabiner, Lawrence R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE 77*(2), 257–285.

Rabiner, Lawrence R. and Biing-Hwang Juang (1993). *Fundamentals of Speech Recognition.* Englewood Cliffs, NJ: Prentice Hall.

Raphael, Christopher (1999a). A Probabilistic Expert System for Automatic Musical Accompaniment. *Jour. of Comp. and Graph. Stats 10*(3), 487–512.

Raphael, Christopher (1999b). Automatic Segmentation of Acoustic Musical Signals Using Hidden Markov Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 21*(4), 360–370.

Raphael, Christopher (2001a). A Bayesian Network for Real Time Music Accompaniment. *Neural Information Processing Systems (NIPS)* (14).

Raphael, Christopher (2001b). A mixed graphical model for rhythmic parsing. In *Proceedings of the 17^{th} Conference on Uncertainty in Artificial Intelligence*, pp. 462–471. Morgan Kaufmann.

Raphael, Christopher (2001c). Music Plus One: A System for Expressive and Flexible Musical Accompaniment. In *Proceedings of the International Computer Music Conference (ICMC)*, Havana, Cuba.

Rational Software (1997, September). Unified Modeling Language, version 1.1. `www.rational.com/uml`.

Recordare (2003, December1). Musicxml definition. Technical report, Recordare Inc. Version 0.9, `http://www.recordare.com/xml.html`.

Rioux, Vincent (2001a). Projet ecrins / validation expérimentale phase 1: descripteurs morphologiques. Technical report, Ircam – Centre Pompidou, Paris, France.

Rioux, Vincent (2001b). Projet ecrins / validation expérimentale problèmatique. Technical report, Ircam – Centre Pompidou, Paris, France.

Rioux, Vincent (2002). Projet ecrins / validation expérimentale 2ème phase: "ergonomie et taxonomie". Technical report, Ircam – Centre Pompidou, Paris, France.

Risset, J.C. and M.V. Mathews (1969, February). Analysis of musical-instrument tones. *Physics Today 22*(2), 23–30.

Roads, Curtis (1988, Summer). Introduction to granular synthesis. *Computer Music Journal 12*(2).

Roads, Curtis (Ed.) (1996). *The Computer Music Tutorial*. Cambridge, Massachusetts: MIT Press.

Rodet, Xavier (1984, Fall). Time-Domain Formant-Wave-Function Synthesis. *Computer Music Journal 8*(3), 9–14. reprinted from (Simon 1980).

Rodet, Xavier (1997a, August). Musical Sound Signals Analysis/Synthesis: Sinusoidal+Residual and Elementary Waveform Models. In *Proceedings of the IEEE Time–Frequency and Time–Scale Workshop (TFTS)*.

Rodet, Xavier (1997b, August). Musical Sound Signals Analysis/Synthesis: Sinusoidal+Residual and Elementary Waveform Models. In *Proceedings of the IEEE Time–Frequency and Time–Scale Workshop (TFTS)*.

Rodet, Xavier (2002, November). Synthesis and processing of the singing voice. In *Proceedings of the 1stIEEE Benelux Workshop on Model based Processing and Coding of Audio (MPCA)*, Leuven, Belgium.

Rodet, Xavier and Boris Doval (1992, October). Maximum likelihood harmonic matching for fundamental frequency estimation. In *124thmeeting of ASA, workshop on "Automatic Tracking of Musical Frequency"*, Volume 92 of *2*, New Orleans, Louisiana, USA. Journal of the Acoustical Society of America (JASA).

Rodet, Xavier and Dominique François (1996, January). *XSPECT: Introduction*. `http://www.ircam.fr/anasyn/xspect/index-e.html`

Rodet, Xavier, Dominique François, and Guillaume Levy (1996, August). Xspect: a New Motif Signal Visualisation, Analysis and Editing Program. In *Proceedings of the International Computer Music Conference (ICMC)*.

Rodet, Xavier and Florent Jaillet (2001, September). Detection and modeling of fast attack transients. In *Proceedings of the International Computer Music Conference (ICMC)*, Havana, Cuba.

Rodet, Xavier and Adrien Lefèvre (1997, September). The Diphone Program: New Features, new Synthesis Methods and Experience of Musical Use. In *Proceedings of the ICMC*, Tessaloniki, Greece.

Rodet, Xavier, Yves Potard, and Jean-Baptiste Barrière (1984, Fall). The CHANT–Project: From the Synthesis of the Singing Voice to Synthesis in General. *Computer Music Journal 8*(3), 15–31.

Rodet, Xavier, Yves Potard, and Jean-Baptiste Barrière (1985). CHANT: de la synthèse de la voix chantée à la synthèse en général. *Rapports de recherche IRCAM*. .

Rodet, Xavier and Diemo Schwarz (2000). *Spectral Envelopes and Additive+Residual Analysis-Synthesis*. In J. Beauchamp, ed. *The Sound of Music*, unpublished.

Rodet, Xavier and Patrice Tisserand (2001, October). ECRINS: Calcul des descripteurs bas niveaux. Technical report, Ircam – Centre Pompidou, Paris, France.

Rodet, Xavier and Patrice Tisserand (2002, March). Ecrins: Rapport sur l'évolution dynamique. Technical report, Ircam – Centre Pompidou, Paris, France. Revision 1.8.

Roebel, Axel (2003). Transient detection and preservation in the phase vocoder. In *Proceedings of the International Computer Music Conference (ICMC)*, Singapore.

Rossignol, Stéphane (2000, July). *Segmentation et indexation des signaux sonores musicaux*. Ph. D. thesis, Université Paris VI. .

Rossignol, Stéphane, Peter Desain, and Henkjan Honing (2001, September). Refined knowledge-based f0 tracking: Comparing three frequency extraction methods. In *Proceedings of the International Computer Music Conference (ICMC)*, Havana, Cuba.

Rossignol, Stéphane, Xavier Rodet, Joël Soumagne, Jean-Luc Colette, and Philippe Depalle (1998, October). Feature Extraction and Temporal Segmentation of Acoustic Signals. In *Proceedings of the International Computer Music Conference (ICMC)*, Ann Arbor, Michigan, USA.

Rousseaux, Francis and CUIDADO partners (2002a). Cuidado updated specification. Technical report, European Commission.

Rousseaux, Francis and CUIDADO partners (2002b). How can feedback from user groups support the CUIDADO project re-specification process? Technical report, European Commission.

Roweis, Sam (1997, February). Hidden markov models. Retrieved from `http://www.cs.toronto.edu/~roweis/notes/hmm.ps.gz`

Örsten Kärki (2003, August). Système talkapillar. Master's thesis, EFREI, Ircam – Centre Georges Pompidou, Paris, France. Rapport de stage.

Sagisaka, Yoshinori (1988). Speech synthesis by rule using an optimal selection of non-uniform synthesis units. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, New York, pp. 679–682.

Schaeffer, Pierre (1966). *Traité des objets musicaux* (1st ed.). Paris, France: Éditions du Seuil.

Schaeffer, Pierre and Guy Reibel (1967). *Solfège de l'objet sonore*. Paris, France: ORTF. Reedited as (Schaeffer and Reibel 1998).

Schaeffer, Pierre and Guy Reibel (1998). *Solfège de l'objet sonore*. Paris, France: INA Publications–GRM. Reedition on 3 CDs with booklet of (Schaeffer and Reibel 1967).

Scheirer, Eric (1998). Tempo and beat analysis of acoustic musical signals. *Journal of the Acoustical Society of America (JASA)* (50), 588–601.

Schwarz, Diemo (1998, June). Spectral Envelopes in Sound Analysis and Synthesis. Diplomarbeit Nr. 1622, Universität Stuttgart, Fakultät Informatik, Stuttgart, Germany. .

Schwarz, Diemo (2000, December). A System for Data-Driven Concatenative Sound Synthesis. In *Proceedings of the COST-G6 Conference on Digital Audio Effects (DAFx)*, Verona, Italy, pp. 97–102.

Schwarz, Diemo (2001, October). IRCAM SDIF Library Tutorial. Web page. `http://www.ircam.fr/sdif/extern/tutorial-main.html`

Schwarz, Diemo (2003a, October). New Developments in Data-Driven Concatenative Sound Synthesis. In *Proceedings of the International Computer Music Conference (ICMC)*, Singapore.

Schwarz, Diemo (2003b, December). Requirements for music notation regarding music-to-score alignment and score following. Technical report, MPEG Ad Hoc Group on Music Notation Requirements. `http://www.dsi.unifi.it/~nesi/mpeg/mn-req-alignment.pdf`.

Schwarz, Diemo (2003c, September). The Caterpillar System for Data-Driven Concatenative Sound Synthesis. In *Proceedings of the COST-G6 Conference on Digital Audio Effects (DAFx)*, London, UK.

Schwarz, Diemo and Nicola Orio (2002, December). Project report score following. Technical report, Ircam, Paris, France.

Schwarz, Diemo and Xavier Rodet (1999, October). Spectral Envelope Estimation and Representation for Sound Analysis-Synthesis. In *Proceedings of the International Computer Music Conference (ICMC)*, Beijing, China. .

Schwarz, Diemo and Matthew Wright (2000, August). Extensions and Applications of the SDIF Sound Description Interchange Format. In *Proceedings of the International Computer Music Conference (ICMC)*, Berlin, Germany, pp. 481–484. .

Schwob, Pierre R. (2003). The Classical Music Archives. Web site. `http://www.classicalarchives.com`.

Selfridge-Field, Eleanor (Ed.) (1997). *Beyond Midi: The Handbook of Musical Codes*. Cambridge, Massachusetts, USA: MIT Press.

Serafin, Stefania and Julius O. Smith (2000, December). A Multirate, Finite-width, Bow–String Interaction Model. In *Proceedings of the COST-G6 Conference on Digital Audio Effects (DAFx)*, Verona, Italy, pp. 207–210.

Serra, X., J. Bonada, P. Herrera, and R. Loureiro (1997). Integrating Complementary Spectral Models in the Design of a Musical Synthesizer. In *Proceedings of the International Computer Music Conference (ICMC)*, Tessaloniki.

Serra, X. and J. Smith (1990). Spectral Modeling Synthesis: a Sound Analysis/Synthesis System Based on a Deterministic plus Stochastic Decomposition. *Computer Music Journal 14*(4), 12–24.

Shalev-Shwartz, Shai, Shlomo Dubnov, Nir Friedman, and Yoram Singer (2002). Robust temporal and spectral modeling for query by melody. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 331–338. ACM Press.

Shonle, J. I. and K. E. Horan (1980). The pitch of vibrato tones. *Journal of the Acoustic Society of America 67*(1), 246–252.

Simon, J. C. (Ed.) (1980). *Spoken Language Generation and Understanding*. Dordrecht, Holland: D. Reidel Publishing Company.

Smith, Julius O. (2003, August). Recent Developments in Musical Sound Synthesis Based on a Physical Model. In *Stockholm Musical Acoustics Conference (SMAC-03)*, Stockholm, Sweden.

Soulez, Ferréol, Xavier Rodet, and Diemo Schwarz (2003, October). Improving Polyphonic and Poly-Instrumental Music to Score Alignment. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, Baltimore, Maryland, USA.

Spevak, Christian (2001, October). Sound Spotting — a Frame-Based Approach. In *2^{nd} Annual International Symposium on Music Information Retrieval (ISMIR)*, Bloomington, Indiana, USA.

Spevak, Christian and Emmanuel Favreau (2002, September). Soundspotter — A Prototype System for Content-based Audio Retrieval. In *Proceedings of the COST-G6 Conference on Digital Audio Effects (DAFx)*, Hamburg, Germany, pp. 27–32.

Sproat, Richard, Paul Taylor, Michael Tanenblatt, and Amy Isard (1997, September). A markup language for text-to-speech synthesis. In *Proceedings of the European Conference on Speech Communication and Technology (EUROSPEECH)*, Rhodes, Greece, pp. 1747–1750. .

Stylianou, Yannis (1996, January). *Harmonic plus Noise Models for Speech, combined with Statistical Methods, for Speech and Speaker Modification*. Ph. D. thesis, Ecole Nationale Supèrieure des Télécommunications, Paris, France.

Stylianou, Yannis (1998a, November). Concatenative Speech Synthesis using a Harmonic plus Noise Model. In *The 3rd ESCA/COCOSDA Workshop on Speech Synthesis*, Jenolan Caves, Australia. .

Stylianou, Yannis (1998b, November). Removing Phase Mismatches in Concatenative Speech Synthesis. In *The 3rd ESCA/COCOSDA Workshop on Speech Synthesis*, Jenolan Caves, Australia. .

Stylianou, Yannis, Thierry Dutoit, and Juergen Schroeter (1997, September). Diphone concatenation using a harmonic plus noise model of speech. In *Proceedings of the European Conference on Speech Communication and Technology (EUROSPEECH)*, Rhodes, Greece, pp. 613–616. .

Stylianou, Y., J. Laroche, and E. Moulines (1995). High Quality Speech Modification based on a Harmonic+Noise Model. In *Proceedings of the European Conference on Speech Communication and Technology (EUROSPEECH)*.

Sundberg, J. (1987). *The Science of the Singing Voice*. Dekalb, Illinois, USA: Northern Illinois University Press.

Syrdal, Ann K, Yannis G Stylianou, Laurie F Garrison, Alistair Conkie, and Juergen Schroeter (1998). Td-psola versus harmonic plus noise model in diphone based speech synthesis. In *Proc. ICASSP98*, pp. 273–276. .

Takala, Tapio, Jarmo Hiipakka, Mikael Laurson, and Vesa Välimäki (2000). An Expressive Synthesis Model for Bowed String Instruments. In *Proceedings of the International Computer Music Conference (ICMC)*, Berlin. ICMA.

Taylor, Paul (1999). The Festival Speech Architecture. Web page, Human Communication Research Centre, University of Edinburgh. .

The PostgreSQL Global Development Group (2002a). *PostgreSQL 7.3 Documentation*. The PostgreSQL Global Development Group.
`http://www.postgresql.org/docs/7.3/interactive/index.html`

The PostgreSQL Global Development Group (2002b). *PostgreSQL 7.3 Programmer's Guide*. The PostgreSQL Global Development Group.
`http://www.postgresql.org/docs/7.3/interactive/programmer.html`

Thom, D., H. Purnhagen, S. Pfeiffer, and the MPEG Audio Subgroup (1999, December). MPEG Audio FAQ. web page. International Organisation for Standardisation, Organisation Internationale de Normalisation, ISO/IEC JTC1/SC29/WG11, N3084, Coding of Moving Pictures and Audio, `http://www.tnt.uni-hannover.de/project/mpeg/audio/faq`.

Tokuda, K., H. Zen, and A. Black (2002). An hmm-based speech synthesis system applied to english. In *IEEE TTS Workshop*, Santa Monica, CA, USA.

Truchet, Charlotte, Carlos Agon, Gérard Assayag, and Philippe Codognet (2001). Cao et contraintes. In *JIM*.

Truchet, Charlotte, Carlos Agon, and Philippe Codognet (2001). A constraint programming system for music composition, preliminary results. In *CP 01, Workshop on Musical Constraints*, Cyprus.

Truchet, Charlotte, Gérard Assayag, and Philippe Codognet (2001). Visual and adaptive constraint programming in music. In *Proceedings of the International Computer Music Conference (ICMC)*, Havana, Cuba.

Turetsky, Robert (2003). MIDIAlign: You did *what* with MIDI? Web page
`http://www.ee.columbia.edu/~rob/midialign`

Turetsky, Robert J. and Daniel P.W. Ellis (2003, October). Ground-Truth Transcriptions of Real Music from Force-Aligned MIDI Syntheses. In *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, Baltimore, Maryland, USA.

Tzanetakis, George (2003, September). MUSESCAPE: An interactive content-aware music browser. In *Proceedings of the COST-G6 Conference on Digital Audio Effects (DAFx)*, London, UK.

Tzanetakis, George, Georg Essl, and Perry Cook (2002, September). Human Perception and Computer Extraction of Musical Beat Strength. In *Proceedings of the COST-G6 Conference on Digital Audio Effects (DAFx)*, Hamburg, Germany, pp. 257–261.

Uneson, Marcus (2002, sep). Outlines of Burcas, a Simple Concatenation-Based MIDI-to-Singing Voice Synthesis System. In *Fonetik*, Stockholm, Sweden.

Uneson, Marcus (2003, jan). Burcas — a Simple Concatenation-Based MIDI-to-Singing Voice Synthesis System for Swedish. Master's thesis, Department of Linguistics, Lund University.

Valbret, H., E. Moulines, and J. P. Tubach (1992, June). Voice transformation using PSOLA technique. *speech 11*(2-3), 189–194.

van Heuven, Vincent J. and Louis C. W. Pols (1993). *Analysis and Synthesis of Speech: Strategic Research towards High-Quality Text-to-Speech Generation*. Berlin, Germany: Mouton de Gruyter.

van Santen, Jan P. H., Richard W. Sproat, Joseph P. Olive, and Julia Hirschberg (Eds.) (1996). *Progress in Speech Synthesis*. New York, USA: Springer-Verlag.

van Vuuren, Sarel and Hynek Hermansky (1997, September). Data-driven design of RASTA-like filters. In *Proceedings of the European Conference on Speech Communication and Technology (EUROSPEECH)*, Rhodes, Greece, pp. 409–412.

Vercoe, Barry (1984). The Synthetic Performer in the Context of Live Performance. In *Proceedings of the International Computer Music Conference (ICMC)*, pp. 199–200.

Vercoe, Barry and Miller Puckette (1985). Synthetic Rehearsal: Training the Synthetic Performer. In *Proceedings of the International Computer Music Conference (ICMC)*, pp. 275–278.

Vincent, E., C. Févotte, and R. Gribonval (2003). Comment évaluer les algorithmes de séparation de sources audio ? In *Proc. 19th GRETSI Symposium on Signal and Image Processing (GRETSI 2003), Paris, France*.

Vincent, E., X. Rodet, A. Röbel, C. Févotte, R. Gribonval, L. Benaroya, and F. Bimbot (2003). A tentative typology of audio source separation tasks. In *Proc. 4th Symposium on Independent Component Analysis and Blind Source Separation (ICA 2003)*, Nara, Japan.

Vinet, Hugues (2003, October16). Les niveaux de représentation de l'information musicale. Atelier "Technologies multimédia et musique" organisé avec le soutien du réseau thématique européen Musicnetwork, Resonances, Ircam.

Vinet, Hugues, Perfecto Herrera, and François Pachet (2002a, Octobre). The CUIDADO Project. In *International Conference on Music Information Retrieval*, Paris, France, pp. 197–203.

Vinet, Hugues, Perfecto Herrera, and François Pachet (2002b, September). The Cuidado Project: New Applications Based on Audio and Music Content Description. In *Proceedings of the International Computer Music Conference (ICMC)*, Gothenburg, Sweden, pp. 450–453.

Virolle, Dominique (1998, January). *Sound Description Interchange Format (SDIF)*. .

Virolle, Dominique, Diemo Schwarz, and Xavier Rodet (2002). SDIF Sound Description Interchange Format. Web page. `http://www.ircam.fr/sdif`

Viterbi, A. J. (1967, April). Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Transactions on Information Theory IT-13*, 260–269.

von Helmholtz, Hermann L. (1913). *Die Lehre von den Tonempfindungen: als physiologische Grundlage für die Theorie der Musik* (6^{th} ed.). Braunschweig: Vieweg. Reprinted as (von Helmholtz 1954; von Helmholtz 1983).

von Helmholtz, Hermann L. (1954). *On the Sensations of Tone as a Physiological Basis for the Theory of Music*. New York: Dover. Original title: (von Helmholtz 1913).

von Helmholtz, Hermann L. (1983). *Die Lehre von den Tonempfindungen: als physiologische Grundlage für die Theorie der Musik*. Hildesheim: Georg Olms Verlag. Original title: (von Helmholtz 1913).

Wakefield, G. H. (1998). Time–Pitch Representations: Acoustic Signal Processing and Auditory Representations. In *Proceedings of the IEEE Intl. Symp. on Time–Frequency/Time–Scale*, Pittsburgh.

Wang, W. J., W. N. Campbell, N. Iwahashi, and Y. Sagisaka (1993). Tree-based unit selection for English speech synthesis. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 191–194.

Wedelmusic (2003). Wedelmusic Home Page. Web page. `http://www.wedelmusic.org`.

Weisstein, Eric W. (2003). Eric Weisstein's World of Mathematics. Web site, Wolfram Research, Inc. `http://mathworld.wolfram.com`.

Wells, John C. (1995, April). Computer-coding the IPA: a proposed extension of SAMPA. Department of Phonetics and Linguistics, University College London, `http://www.phon.ucl.ac.uk/home/sampa/x-sampa.htm`.

Wells, John C. (2003, March). SAMPA for French. Web page. Department of Phonetics and Linguistics, University College London, `http://www.phon.ucl.ac.uk/home/sampa/french.htm`.

Wessel, David, Cyril Drame, and Matthew Wright (1998). Removing the Time Axis from Spectral Model Analysis-Based Additive Synthesis: Neural Networks versus Memory-Based Machine Learning. In *Proceedings of the International Computer Music Conference (ICMC)*, Ann Arbor, Michigan. ICMA. .

Wöhrmann, Rolf and Guillaume Ballet (1999, Fall). Design and architecture of distributed sound processing and database systems for web based computer music applications. *Computer Music Journal 23*(3), 73.

Wouters, J. and M. W. Macon (1998, November). A perceptual evaluation of distance measures for concatenative speech synthesis. In *Proceedings of the International Conference on Spoken Language Processing (ICSLP)*. .

Wright, Matthew, James Beauchamp, Kelly Fitz, Xavier Rodet, Axel Röbel, Xavier Serra, and Greg Wakefield (2000). Analysis/synthesis comparison. *Organised Sound 5*(3), 173–189.

Wright, Matthew, Amar Chaudhary, Adrian Freed, Sami Khoury, Ali Momeni, Diemo Schwarz, and David Wessel (2000). An XML-based SDIF Stream Relationships Language. In *Proceedings of the International Computer Music Conference (ICMC)*, Berlin. .

Wright, Matthew, Amar Chaudhary, Adrian Freed, Sami Khoury, and David Wessel (1999). Audio Applications of the Sound Description Interchange Format Standard. In *AES 107th convention preprint*. .

Wright, Matthew, Amar Chaudhary, Adrian Freed, David Wessel, Xavier Rodet, Dominique Virolle, Rolf Woehrmann, and Xavier Serra (1998). New Applications of the Sound Description Interchange Format. In *Proceedings of the International Computer Music Conference (ICMC)*. .

Wright, Matthew, Richard Dudas, Sami Khoury, Raymond Wang, and David Zicarelli (1999, October). Supporting the Sound Description Interchange Format in the Max/MSP Environment. In *Proceedings of the International Computer Music Conference (ICMC)*, Beijing. .

Wright, Matthew and Eric D. Scheirer (1999, October). Cross-Coding SDIF into MPEG-4 Structured Audio. In *Proceedings of the International Computer Music Conference (ICMC)*, Beijing. .

Yi, Jon and James Glass (2002, September). Information-theoretic criteria for unit selection synthesis. In *Proceedings of the International Conference on Spoken Language Processing (ICSLP)*, Denver, Colorado, pp. 2617–2620.

Yi, Jon Rong-Wei (2003, May). *Corpus-Based Unit Selection for Natural-Sounding Speech Synthesis*. Ph. D. thesis, Massachusetts Institute of Technology.

Yoshimura, T., K. Tokuda, T. Masuko, T. Kobayashi, and T. Kitamura (1999). Simultaneous Modeling of Spectrum, Pitch and Duration in HMM-Based Speech Synthesis. In *Proceedings of the European Conference on Speech Communication and Technology (EUROSPEECH)*, pp. 2347–2350.

Zils, Aymeric and François Pachet (2001, December). Musical Mosaicing. In *Proceedings of the COST-G6 Conference on Digital Audio Effects (DAFx)*, Limerick, Ireland.

Zils, Aymeric and François Pachet (2003, September). Extracting automatically the perceived intensity of music titles. In *Digital Audio Effects (DAFx)*, London, UK.

Zwicker, Eberhard (1982). *Psychoakustik*. Berlin, Germany: Springer Verlag.

Index