Composing with EncycloSpace: a Recombinant Sample-based Algorithmic Composition Framework

Yury Viacheslavovich Spitsyn

Moscow, Russian Federation

A Dissertation presented to the Graduate Faculty

of the University of Virginia in Candidacy for the Degree of

Doctor of Philosophy

Department of Music

Advisor: Professor Judith Shatin, Ph.D.

University of Virginia

July 31, 2015

Abstract

Recorded sounds have been in compositional circulation since the early days of *musique concrète*, yet there have been major lacunae in the design of computer tools for sample-based algorithmic composition. These are revealed by the absence of algorithmic methods based on quantified representation of the sampled sounds content. With the exception of *pitch-tracking* and *amplitude following*, the parametric utilization of recorded sounds lags far behind that of synthetic sound methods, which are parametric by design.

During the last decade, musical informatics have established computational methods to break through the informational opaqueness of sound recordings. By extracting semantic data (called *descriptors* or *features*), sampled sounds can be converted into quantifiable form and more easily described, diagrammed and compared. This technology has resulted in numerous compositional applications focused on descriptor-based sound manipulations and database usage. Surprisingly, compositional strategies have gravitated to a narrow range of manipulations, called *concatenative synthesis* and *music mosaicing*. Inherent limitations in current approaches motivated a search for open-ended design principles for corpus-based sample-oriented algorithmic composition.

This dissertation examines theoretical and practical ways to approach this problem before proposing a solution. To avoid conceptual shortcomings, we start with an overarching perspective by introducing the ontological topology of music as developed by Guerino Mazzola. We use his comprehensive analysis of activities, scientific disciplines, ontological dimensions and informational processes in music as a basis for our approach to system design in a balanced and deliberate way. Mazzola's notion of EncycloSpace, the knowledge base of musical process, and the concepts of *receptive* and *productive* navigation serve as a conceptual core for the design. The overall system's structure is tripartite: an *analytic subsystem*, a *compositional subsystem*

and EncycloSpace — *database subsystem*.

A methodology, termed *NOTA-transform*, is proposed as a general method for the compositional use of EncycloSpace. It defines four consecutive transformations, called *navigation, ordering*, *temporalization* and *adaptation*, as indispensable stages of converting EncycloSpace into a musical object. Each of the stages is defined as an independent decision-making site. With respect to *ordering* and *temporalization*, most of the existing corpus-based algorithmic systems are either deterministic or indifferent. The system design considerations are then used to implement a working software prototype called *sEl*, described in detail in Chapter 3.

Finally, sEl is utilized to compose a piece, *Ignis Fatuus* (4), where I demonstrate the resulting analytic and compositional algorithms.

Acknowledgements

As I complete this dissertation, I am thankful to a great many people, whose diverse contribution made this work possible (and much more enjoyable than it would be otherwise).

I am grateful foremost to my thesis advisor, Judith Shatin, for her support, encouragement and guidance throughout this lengthy process. Her thoughtful comments and profound editorial help were truly invaluable. Had she not cared about this work's progress as much as she did, it might still be in an unfinished state.

Thanks to my dissertation committee members, Ted Coffey, Luke Dahl and Worthy Martin, for their time, interest and expertise in discussing/reviewing this project.

I deeply appreciate the UVa Music Department for being an open-minded collective of music scholars and practitioners where wide range of research topics is encouraged.

A very special thanks to Bonnie Gordon and Kyle Ruempler at the McIntire Department of Music for their incredible administrative support, especially during the last year, when it was the most needed.

I'd like to express my deepest gratitude to Jon Appleton for being a long-time supporter and friend, which is very humbling. Among the innumerable occasions of support, thanks for providing sanctuary where I could go on with my writing during its most critical phase. Snowy winters in Vermont – my favorite season – have been a wonderful place to concentrate without interruption.

Much gratitude to the people whom I was fortunate to meet and who, throughout the years, influenced my understanding of all things musical through the meetings, seminars, conversations and collaborations: Paul Bothello, Matthew Burtner, Michael Casey, Ted Coffey, Nick Collins, Perry Cook, Roger Dannenberg, Charles Dodge, Kui Dong, David Dunn, I-Jen Fang, Seppo Gruendler, Franz Hautzinger, Kurt Hebel, Jaroslaw Kapuscinski, Lydia Kavina, Tatiana Komarova, Michael Kubovy, Paul Lan-

sky, Sergey Letov, Ted Levin, Lukas Ligeti, Victor Luferov, Eric Lyon, Max Mathews, Paula Matthusen, Fred Maus, Gordon Monahan, Vladimir Nikolaev, Tae Hong Park, Larry Polansky, Godfried-Willem Raes, Douglas Repetto, Paul Riker, Jean-Claude Risset, Carla Scaletti, Elizabeth Schimana, Judith Shatin, Daniel Shorno, Andrey Smirnov, Tom Stoll, James Tenney, Trimpin, Dave Topper, Dmitri Ukhov, Kojiro Umezaki, Yiorgos Vassilandonakis, Barry Vercoe, German Vinogradov, Ge Wang, Rebekah Wilson (aka Netochka Nezvanova), Christian Wolff, Maurice Wright and Sergey Zagny.

Thanks to the outstanding music graduate student communities at Dartmouth, where I worked, and at UVa, where I studied, and with whom I shared a perennial appetite for musical matters. These include Scott Barton, Courtney Brown, Carmen Caruso, Kevin Davis, Erik DeLuca, Charlie DeTar, Carlos Dominguez, Irina Escalante-Chernova, Travis Garrison, Sarah O'Halloran, Will Haslett, Aurie Hsu, Wendi Hsu, Kyle Kaplan, Angela Kim, Steve Kemper, Juraj Kois, Masaki Kubo, Liz Lindau, Loren Ludwig, Ryan Maguire, Owen Osborn, Kevin Parks, Chris Peck, Sean Pequet, Brent Reidy, Troy Rogers, Bruno Ruviaro, Lanier Sammons, Andy Sarroff, Danny Shapira, Victor Shepardson, Braxton Sherouse, Beau Sievers, Ezra Teboul, Peter Traub, Paul Turowski, Shannon Werle and Jonathon Zorn.

Special thanks to the open-source software community to which I am immensely indebted.

An electrically-charged thanks goes to Ezra Teboul and Kyle Kaplan for the diligence and fun of making it together through the last year in one piece.

Thanks to Jodie Mack for being great and always making it more interesting.

Two Charlottesville cats, Ami and Jemma, deserve to be mentioned for being my only companions (as well as occasional furry combat entertainment) for six months. Thanks to Jane and Bruce Penner for arranging things that way.

Chirpful thanks to the Vermont birds for being a harmonious summer-time accompaniment to my writing.

Thanks to David Vincelette for his help with ceiling-mount of eight speakers in my room and to Anna Vincelette for those frequent and tasty meals.

Thanks to my friends and colleagues in Russia who, after all these years of my not being present, still remember my name.

Lastly, my undying appreciation to my family: my mother Alevtina Spitsyna and my son Nikita Spitsyn, who watched my multi-year involvement in this project with understanding and love. My great gratitude to Lorina Kaydanovskaya for her incredible feat of patience and belief in me.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

"Music is a research the result of which is music."

Unknown source

Music creativity, technology and music theory have been intertwined and have increasingly influenced each other for centuries. From the standpoint of music creation, the breakthroughs in music technology and theoretical thought have stimulated new ideas and brought them under the spotlight of compositional research. The motivation for this dissertation is no different — we are seeking to develop a computer tool for algorithmic composition that uses theoretical frameworks and recent music technologies to try to settle on a territory that can potentially expand contemporary compositional practice. This territory is associated with methodologies based on the algorithmic treatment of sampled sound materials guided by the usage of semantics extracted from the materials. In the history of computer-aided algorithmic compositional (CAAC) software there has been a long-standing paucity in that area because the aforementioned methodologies require technological tools which, until recently, have not been widely available.

The theoretical basis of our project is supported by the research in music ontology and its technical grounding is based on the recent technologies developed in the field

of musical informatics. The importance of both will be discussed in this introductory chapter along with specific problematics of sample-based algorithmic composition.

1.1 Music ontology and EncycloSpace

The process of music creation is an intricate ontological tapestry of decisions, actions and states, both objective and subjective. Reliance on music ontology equips one with the theoretical apparatus suitable for any music-related task, including designing compositional tools; it will also help to locate focal design areas and to understand their relations within the larger context. A clear ontological taxonomy can also be an indispensable tool for the disambiguation of semantics within the sophisticated non-linearities of the music creation process, about which Horacio Vaggione writes in [118]:

> This task [composing] cannot be exhausted by a linear (a priori, non-interactive) problem-solving approach. Interaction is here matching an important feature of musical composition processes, giving room for the emergence of irreducible situations through non-linear interaction.

Our design principles are inspired by the ontological schema presented in Guerino Mazzola's comprehensive treatise on music topology [76]. Since its concepts of *EncycloSpace* and ontological dimensions of music are used throughout this work we will briefly expound its core terminology in the following sections.

1.1.1 Fundamental activities of music

Mazzola defines four fundamental activity types that pertain to music, that of *production, reception, documentation* and *communication.*

Production refers to any generative type of activity related to music making.

Reception encompasses activities by any agency, human or not, involved in the evaluation of music.

Documentation refers to activities that attribute to music entities a state of permanency, such as symbolic notation, media recordings, database storage etc.

Communication represents activities involved in communication among the processes of *production, reception* and *documentation*.

1.1.2 Fundamental scientific domains

Along with these core areas Mazzola names a set of *fundamental scientific domains* related to the field of music and defined as follows ([76], p.6):

> Now, if one has to locate a special research subject, such as music, within the hierarchy of disciplines, one is looking for a minimal set, a kind of basis of disciplines which are necessary to cover the subject. In this sense, we argue that four sciences: semiotics, physics, mathematics, and psychology, are such a basis.

This roster of scientific domains strongly supports the notion of music as a multidisciplinary field which, to adequately cover its meaning, requires the presence of multiple descriptive languages.

Semiotics deals with music as a sign system, considering contexts such as symbolic notation, referential properties of sound as a mimetic device, cultural referencing etc.

Mathematics is involved to describe the complex forms that music takes as a system of proportions, as a temporal process and any other quantifiable aspect.

Physics covers descriptions of music processes on the level of physical reality.

Psychology deals with phenomenological aspects of music, such as perception and cognition, musicological interpretation, culturological assessment etc. Fundamental activities and scientific domains can be mutually associated as shown in figure 1.1.

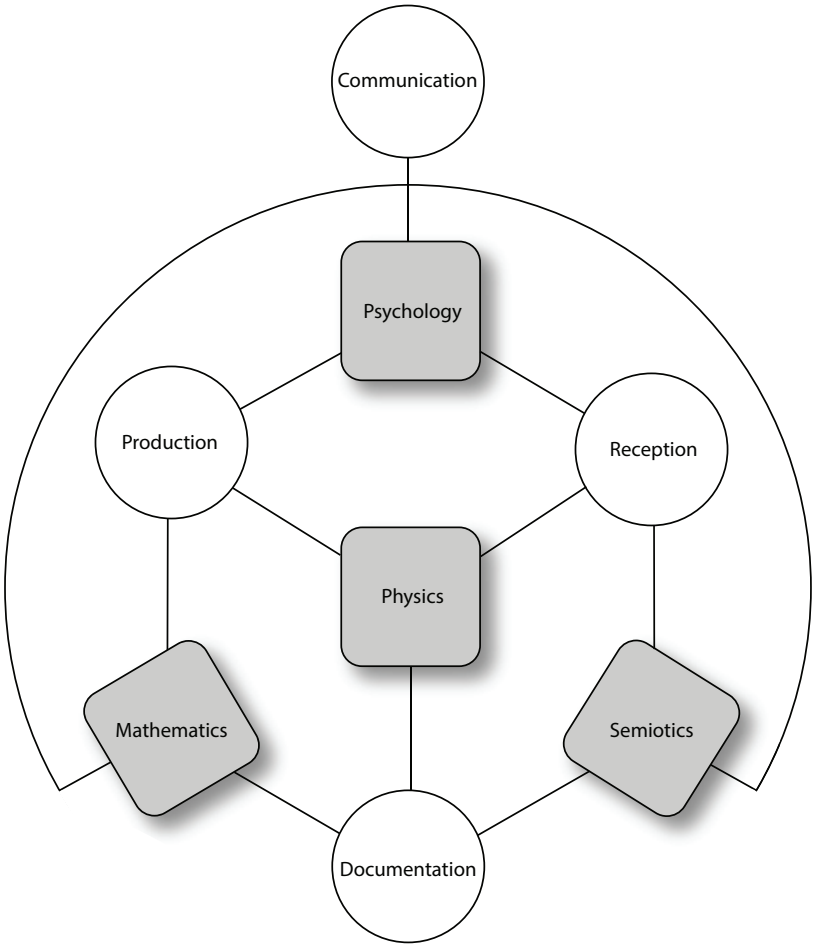


Figure 1.1: Association graph of fundamental activities and scientific domains of music

1.1.3 Coordinate space of ontological dimensions

Mazzola approaches music ontology in a topographical sense — instead of defining ontological categories as a set of spaceless determinants he first introduces three

ontological dimensions, *Reality, Communication* and *Semiosis*, as a coordinate basis for holding any fact of music ([76], p.10):

> To understand music as a whole, you have to specify simultaneously its levels of reality, its semiotic character, and its communicative extension. Being a fact of music means having these three perspectives or ontological coordinates. Omitting any of these determinants is an abstraction ... from the full ontology.

Each ontological dimension is further topologically subdivided onto three autonomous layers. These will be briefly described in the following sections as we are using that terminology in our work.

1.1.3.1 Reality

The substance of music occupies reality on a range of layers: *physical, psychological* and *mental*. According to Mazzola, none of these can be eliminated without inducing reductionist view. It is also not "ontologically possible to reduce one reality to others" ([76], p.10), because music phenomena have different representation in each. Instead, the problem is to describe the correspondence between representations in different layers of reality.

Physical layer of reality has to do with music as a physical temporal process: vibration of physical bodies, such as strings, membranes; radiation of an acoustic wave through the acoustic medium, physical disturbance of cochlea, physiological propagation via an auditory nerve etc.

Mental layer of reality deals with music entities realized as mental schemas, projections of musical ideas into physical reality: symbolic scores, algorithmic formalisms, analytic reductions such as Schenkerian diagrams etc.

Psychological layer of reality includes aspects of music related to its ability to create emotional response in the listener. It is a substantive and autonomous reality, for the very same entities of physical and mental layers can produce very different emotional responses on the psychological level.

1.1.3.2 Communication

The *communication* dimension's definition is drawn from the music theory laid out by Jean Molino in [83] and elaborated by Jean-Jacques Nattiez in [84]. The theory defines music as a tripartite communication involving *creative (poietic), neutral (trace)* and *receptive (esthesic)*[1] components.

Emittance layer (poiesis) is associated with the combination of factors leading to creation of a musical work; the poiesis layer involves any agency, human or automated, that partakes in building the music "message".

Trace layer (neutral) refers to the ontological dimension of the music work itself. With all the difficulties of identifying what exactly the music work is, its ontological aspect does not require pinpoint precision in the definition or, as Molino puts it, "the idea is intuitively clear" ([83], p.152). The *neutral* layer defines the music work as a *trace* produced by the *poiesis* layer and available for interpretation (re-construction of trace) by the *esthesis* layer.

Reception layer (esthesis) refers to the receiving end of the communication schema which is — the listener. The term *listener* should be taken in a general sense — as a combination of factors involved in the valuation of music facts. For instance, with respect to communication, a human listener and a music information retrieval algorithm will be located at similar ontological coordinates.

[1]Notice special spelling in *poietic* and *esthesic* to divorce their meaning from the conventional *poetic* and *aesthetic*

1.1.3.3 Semiosis

Involving intricate processes of signification, music can be characterized as "one of the most developed non-linguistic systems of signs" ([76], p.16). Signification takes places on various levels of *reality* and *communication*, from symbolic notation to production/reception of musical facts. Mazzola constructs the dimension of music semiosis borrowing from the structuralist semiology as developed by F. de Saussure in [38] and R. Barthes in [16]. It should be noted that applying semiological principles to music does not entail attributing to it any properties of a language. A detailed discussion of semiological aspects of music would go well beyond the scope of this work; suffice is to say that musical meaning grows from very complex trajectories of signification. With its spatio-temporal syntax, music does not convey clear and unambiguous meaning; instead, it creates endless threads for emotional and mental interpretation.

The dimension of *semiosis* has a tripartite layout corresponding to the components of a *sign* realization in the structuralist semiology: *signification, signifier* and *signified*.

Signification is the act of using one entity to point to another. A simple example: a ***sfz*** sign in Western notation points to the idea of playing sound with an accent. During the *signification* act, a translation of the sign takes place that dereferences its content in the form of meaning.

Signified is a content of *signification* pointed at by the *signifier*.

Signifier is a component of *signification* that has to be interpreted in order to obtain meaning contained in *signified*. *Signifier* itself can become a complete *sign* thus creating possibilities for an infinite chain of cascading interpretations.

1.1.4 Music ontology cube

Taking *reality, communication* and *semiosis* as an orthogonal basis, Mazzola defines a finite three-dimensional space where each dimension is represented by discrete values corresponding to the layers described above. Since there are three dimensions and each one has three layers, there are $3^3 = 27$ ontological coordinates collectively forming a cube (see figure 1.2). Each coordinate is ontologically distinct and is not reducible to any other combination of coordinates. Each coordinate represents one of 27 ontological positions that particular music object can occupy and which is defined as a combination of locations in *reality, communication* and *semiosis*.

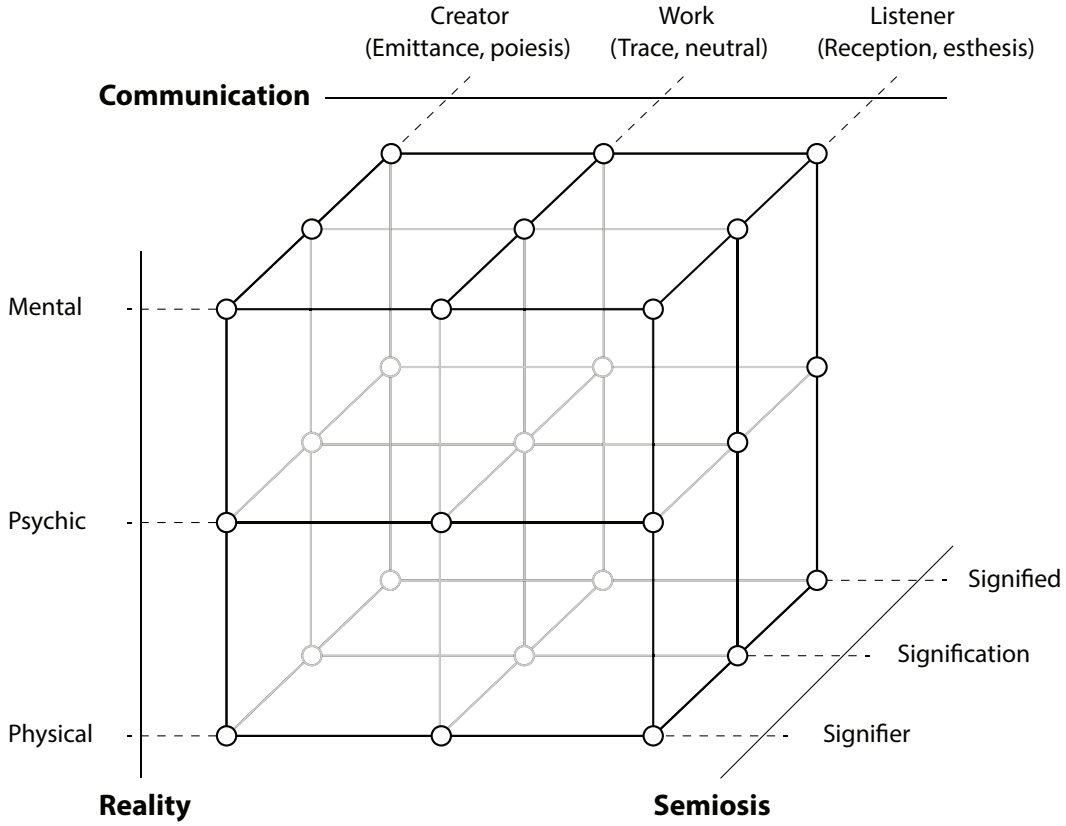


Figure 1.2: The cube of music ontology

An important property of the ontology cube is its local orientation and possibility of recursion — each of its nodes can yield a whole cube for further navigation which

Mazzola terms as a "conceptual zoom-in effect" ([76], p.21). Consequently, ontological investigation about the fact of music is likely to form a path of ramifications through the nodes of the cube.

1.1.5 EncycloSpace

Musical facts are not created and perceived in an informational vacuum. The concept of *EncycloSpace*, defined in [76], postulates that a musical process takes place in incessant communication with a dynamically changing body of knowledge. Stressing the fact that knowledge has two parts, "information and its mental organization", Mazzola points out the role of conceptually diverse arrangements of information, crucial to the effectiveness of its access. In that sense, EncycloSpace is a conceptually diverse, organized body of information which allows the optimized search for data. Another crucial EncycloSpace's characteristic is its dynamicity — in the presence of newly appearing musical facts it constantly updates its informational volume and its conceptual organization. The dynamicity of EncycloSpace leads to a distinction between two counter-directional modes of its access: *Receptive* and *Productive*.

1.1.5.1 Receptive navigation

Receptive navigation corresponds to the method of access typical in traditional encyclopedias — when information is retrieved without altering the knowledge base. The immutable character of receptive navigation reflects the *esthesic* dimension of music ontology.

1.1.5.2 Productive navigation

In contrast with the previous, *Productive navigation* entails an accretion of information in EncycloSpace and possibly, also an extension of its conceptual vocabulary. This navigation mode is responsible for the EncyloSpace's dynamicity.

1.2 Problematics of sample-based algorithmic composition tools

In this section we are going to argue that, until recently, sample-based algorithmic composition tools have not been in active development (and therefore lag behind other automated composition computer tools) due to the intrinsic conflict between the requirements of EncycloSpace access and the parametrically opaque nature of sampled sounds.

Algorithmic systems of composition vary in the ways in which they approach the formalization of decision-making strategies. As part of the strategy, most of them have to deal with *generative* and *evaluative* parts of the compositional cycle [109], which, in the above ontological terminology, belong to *productive* and *receptive* dimensions correspondingly. Generative algorithmization is focused on productive strategies, while evaluative formalisms are mainly concerned with the assessment of generated structures in order to decide on their fitness for the inclusion in the final result. More often than not, both generative and evaluative strategies are realized as parametric systems endowed with elaborated semantical structures. In ontological terms, both *poietic* and *esthesic* formalisms serve as parts of EncycloSpace in which their semantic structure defines the conceptual organization for algorithmic access, while their parameter values provide information which is being accessed and used for the creation of music. This brings to the forefront an important consequence of switching from traditional to algorithmic methods of composition: computer formalisms should be capable of supporting the system's EncycloSpace navigation, preferably in both, *receptive* and *productive*, modes of access. Put another way, delegating compositional decision-making to algorithms comes with the necessity of algorithmic maintenance of the knowledge base of the system.

EncycloSpace navigation, being an intrinsic requirement for algorithmic implementation, has always been an obstacle for the realization of sample-based compositional strategies due to the parametric opaqueness of sampled sound. On the contrary, synthetic methods of sound computation never experienced similar problems due to their inherently parametric nature and, therefore, effortless integration into algorithmic EncycloSpace. This difference, in our opinion, is responsible for the prevalence of algorithmic systems whose compositional vocabulary is predominantly based either on synthetic elements or on symbolic sign systems such as MIDI or notation.

Despite the parametric numbness of sampled sounds, their natural complexity and unrefined character guaranteed their longevity as choice compositional material since the dawn of *musique concrète*, although chiefly, in non-algorithmic contexts. Whenever I used sampled materials in my algorithmic works there has always been a great deal of manual labor combined with randomized material navigation — obviously very limiting and an unsatisfying choice of strategy. An exception to the rule is *granular synthesis*, which allows treatment of synthesized and recorded material on an equal footing providing comparable levels of parameterization for both. Still, this is not a general solution, because granular synthesis parameters, in the case of samples, are agnostic to the grain's content.

The situation with the algorithmic comprehension of sampled signals started to change due to the technological advances in the field of speech synthesis in the middle of 1990s [101]. Concatenation-based speech synthesis relied on large databases of recorded speech elements, which had to be extensively parameterized in order to be navigable. Over the last decade, technologies have emerged that brought recorded sound data mining on the level of widespread availability in the form of commercial and open-source music information retrieval toolsets. This technological surge en-

abled the new design of computer tools for sample-based algorithmic composition. Indeed, numerous applications were developed that utilize sound production via navigation of the sample-oriented EncycloSpace. This methodology of creating music facts was termed *corpus-based concatenative synthesis*. Discussion in [112] and [101] contains overviews and a description of strategies of a number of corpus-based composition systems. The description shows that many of those systems are mostly designed as synthesis tools, i.e. their compositional strategies are rather narrowly designed and confined within the predetermined decision-making procedures. These problematics will be addressed in more detail in 2.5.2.

The overall assessment of the state of corpus-based algorithmic compositional systems, drawn from the above and other publications, shows that the need for a general approach and balanced methodology in that area is still required. In the main, our goal was to design and implement a computer system for algorithmic composition which meets the following objectives:

- System design should include the concepts of musical ontology and EncycloSpace (*corpus-based*)
- A general methodology of compositional use of EncycloSpace should be devised
- EncycloSpace should support recorded material of arbitrary provenance (*omni-source*)
- The system should support *receptive* and *productive* modes of the EncycloSpace navigation
- The system should support semiotic signification within EncycloSpace (*recombinance*)
- The system should provide full production cycle — from compositional design to perceivable musical result.

In the following chapter the system design will be discussed in detail.

Chapter 2

System design

This chapter discusses structural and functional considerations for building compositional software based on the intended functionality (1.2). We will first examine basic aspects such as system identification, choice of development platform, system dependencies etc. The following specialized sections discuss the system architecture, media database considerations and approaches to specific parts of the system.

2.1 General considerations

2.1.1 System identification

This software's functional identification can be constructed from CAAC taxonomy[1]. This taxonomy is not yet standardized and has a history of multiple development stages. An attempt to systematize computer music tools, proposed in [93] in the middle of 1990s, places *Algorithmic and computer-aided composition* tools in the section *Music theory, composition and performance*. There are only three subdivisions in that taxonomy:

1. Compositional algorithms and languages

[1]Computer Aided Algorithmic Composition

2. Composition systems for score or sound synthesis
3. Artificial intelligence and composition

Divisions like this seem too general and not necessarily based on well-defined principles of categorization. A much more systematic approach was suggested in [12]:

> In order to describe the landscape of software systems, it is necessary to establish distinctions. Rather than focusing on chronology, algorithms, or output types, seven descriptors of CAAC system design are proposed. These descriptors are scale, process time, idiom affinity, extensibility, event production, sound source, and user environment. All systems can, in some fashion, be defined by these descriptors.

According to these seven descriptors our CAAC system can be characterized as follows:

2.1.1.1 Scale

The scale descriptor defines the level of temporal structures that the system can produce. It changes from *micro* (sample level, grains, notes) to *macro* (sections, movements, whole compositions). This range comprises five of nine time scales defined in [97], namely *sample, micro, sound object, mesa, macro.*
With respect to the scale descriptor our project can be characterized as a *Full-scale system* — it produces structures of all scales from micro objects to large-scale structures.

2.1.1.2 Process time

The process time descriptor assesses systems with respect to real-time operation, thus differentiating between real-time (RT) and non-real-time (NRT) architectures.

Real-time operation means that in all relevant situations the system guarantees computation of the audio frame in less time than it takes to play it back as sound. Normally, this is a welcome property and more often than not computer music systems come equipped with it. Nevertheless, in select contexts real-time operation can become a bottleneck for the functionality of the system. An example of such a context is a situation when audio production depends on the analysis of incoming sound. In this case, there will always be delays correlated with the analysis' temporal parameters. For instance, the rhythm transform can utilize analysis frames up to 6 seconds [55]. In such situations, real-time requirements should be lifted or they start conflicting with computation of the sound data[34]. More examples of relaxed real-time sound application contexts can be found in [104]

In the current project, compositional computation essentially relies on analytic processes. Some production methods require multi-pass procedures to attain the necessary perceptual precision. Stochastic aspects of rendering entail non-linear access to audio buffers which is also incompatible with real-time linearity. While real-time operation is still desirable, currently this requirement is lifted for the benefit of exactitude and quality of the resulting computation.

2.1.1.3 Idiom affinity

Idiom affinity is defined in [12] as

> ...the proximity of a system to a particular musical idiom, style, genre or form. ... A system with a singular idiom affinity specializes in the production of one idiom (or a small collection of related idioms), providing tools designed for the production of music in a certain form, from a specific time or region, or by a specific person or group. A system with a plural idiom affinity allows the production of multiple musical styles, genres, or forms.

Idiom affinity is here defined by its particular model of musical process based on the following principles:

- Receptive/Productive behavior — the compositional process depends on an analytic process.
- EncycloSpace — compositional models utilize semantic space.
- Recombinance — compositional API supports recombinant treatment of material.
- Omni-source — sounds derived from any source, time-scale and character can be processed by the system.

2.1.1.4 Extensibility

Extensibility of the system characterizes its possibility of functional expansion from the user's vantage point. Systems can be expanded in various ways and on various levels. Plug-ins, extension modules, object macros, script engines, unit generators, backends — these are just a few examples of expansion methods. Code-driven systems are expanded by adding code, data-driven systems - by adding new data. Systems designed for extension are called *open*, the opposite are called *closed*. Being open-source, this system allows full code-driven extensibility to users who can program. The analysis subsystem also provides a modular interface to extend the system by adding new analysis algorithms (3.3.10). On the data-driven side, this system supports expansion via productive utilization of EncycloSpace.

2.1.1.5 Event production

The event production descriptor is concerned with ways in which new material is delivered. It discriminates between *generative* and *transformative* systems. In the current implementation, this system mostly offers tools for recombinant transformations of

existing material. However, nothing precludes its future expansion with generative objects.

2.1.1.6 Sound source

This descriptor characterizes how systems deal with sound realization:

- Internal — the sound result is produced by an internal processing engine.
- Exported — the system produces result in some descriptive format that can be converted to sound by an external system
- Imported — ability of the system to import external system definitions and convert them to sonic results
- External — the system produces results which can be only realized externally. The main difference between this and exported modes is the level of the user's control over sound parameters: the exported mode allows such control while the external mode does not.

This system produces sound results using the internal processing engine. It can also import segmentation and analysis data from external sources.

2.1.1.7 User environment

As defined in [12], the user environment discerns compositional systems by the ways in which they present their functionality to the user:

- Specialized Text Languages — programming language extensions and languages specialized for CAAC operation. Examples include LISP-based extensions (e.g. Common Lisp Music[99], Nyquist[37], Extempore[108]), HMSL[27] and JMSL[39], Csound[120], SuperCollider[77], ChucK[121]. RTcmix[51] uses swappable front ends (Minc, Python, Ruby) which allows user to choose from multiple syntaxes.

- Specialized Graphic Languages — visual language environments most notable of which are Max/MSP[94], Pure Data[95], Kyma [30], OpenMusic[10] and PWGL[69].
- Batch processors — software that provides little or no interactivity to the user with the exception of specifying some system parameters before it starts processing. Batch processors are often developed as algorithmic environments whose generative decision-making process is ultimately automated. Examples include early systems by Hiller and Isaacson[59], Koenig[66] and [65], Xenakis[61], Zaripov[124], recombinant systems by Cope[36], some music mosaicing software [125], [21], [58] to name just a few.
- Interactive Text interface systems — text-based interaction systems with more specialized constructs than that of languages. As examples of software that falls into this category could be mentioned compositional systems by composers Barry Truax[114] and Trevor Wishart[45], Bol Processor[18], athenaCL[13] etc.
- Interactive Graphic interface systems — software in which user interaction is administered via graphic elements. Examples include SoundLoom (GUI layer for CDP software)[122], CsoundQt (GUI for Csound)[28], CataRT[102], Nodal[78] and more.

Our system combines features of Interactive Graphic interface systems and Specialized Text Languages.

The full seven-descriptor identification of our system includes:

- scale — *full time-scale*
- process time — *relaxed real time*
- idiom — *recombinant, corpus-based*
- extensibility — *open*
- event production — *transformative* (in the future will add *generative*)

- sound source — *internal* and *imported*
- user environment — *interactive graphic interface*

2.1.2 Development platform and language

Music-oriented software varies in its particular programming languages and platforms. Platform contexts can significantly differ in scope — from operating systems to specific software packages that attain platform capabilities via extension interfaces (usually in the form of plugin APIs). The language and platform choice is informed by functionality requirements, the targeted user base, and by logistics, such as development timeframe, developer resources etc.

2.1.2.1 Platform

Computer-aided composition software can either be designed from scratch using general operating system tools and run as a separate process or it can be hosted by specialized software that provides audio and other functionality thus reducing the amount of development required. Some systems, like Csound ([120]) or RTcmix ([51]), which were originally developed as standalone tools eventually added versions embeddable in specialized platforms like MaxMSP ([94]).

However attractive the hosted development option, it comes with limitations imposed by the hosting software's architecture and conceptual idiosyncrasies. A hybrid approach can be developed in order to avoid such restrictions: the software is developed on a general platform while integrating specialized third-party libraries. This allows a reduction in development time without constraining functional coverage. On the other hand some development overhead can be induced by the necessity of integration tasks.

For this project a hybrid approach was adopted. Another example of an integrated system can be found, for instance, in [50].

2.1.2.2 Programming language

At the current stage, this project's software is a prototype (as opposed to a completed system). For its development we chose the Python language. This decision was informed by the availability of open-source third-party components and tools and by Python's suitability for rapid development. More details regarding preference for Python for development of CAAC environment can be found in [13].

2.1.3 Dependencies

This project integrates the functionality of a number of open-source frameworks and libraries to optimize the development cycle. Their roster is provided in Appendix D.

2.2 System structure

The software is designed as a set of collaborating subsystems each with its scope and function. The structural components can be divided into two groups: a function carrying group and a user interaction (UI) group. Components of a functional group do all the data computation work while UI components assist in accessing and controlling the functional parts. Figure 2.1 shows principal design and channels of communication among system components.

2.2.1 User interface

Software user interfaces range from simple command-line text input to elaborated graphical systems. This project's software manages large amounts of audio and analytic data that would be very unwieldy (if possible at all) to manipulate and understand without graphical tools for navigation, visualization, data selection etc. The project's user interface uses windowing system based on an open-source cross-platform framework called Qt[35]. The user interface is presented in more detail in 3.1.

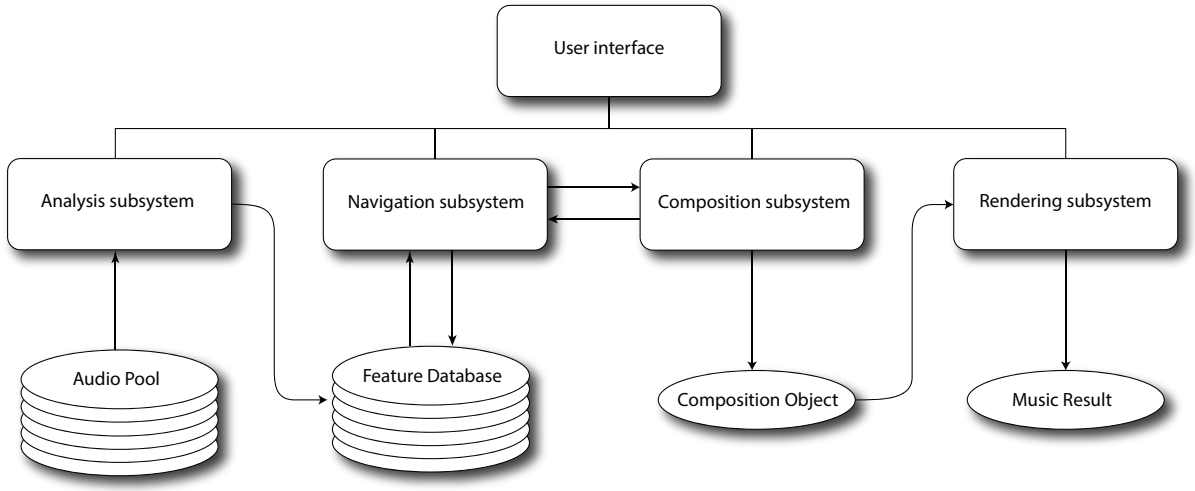


Figure 2.1: System structure

2.2.2 Functional structure

The functional structure of the software follows the tripartite ontological communication scheme of music outlined in 1.1.3.2. In accordance with the schema's layers, the functional parts comprise three communication groups depicted in figure 2.2.

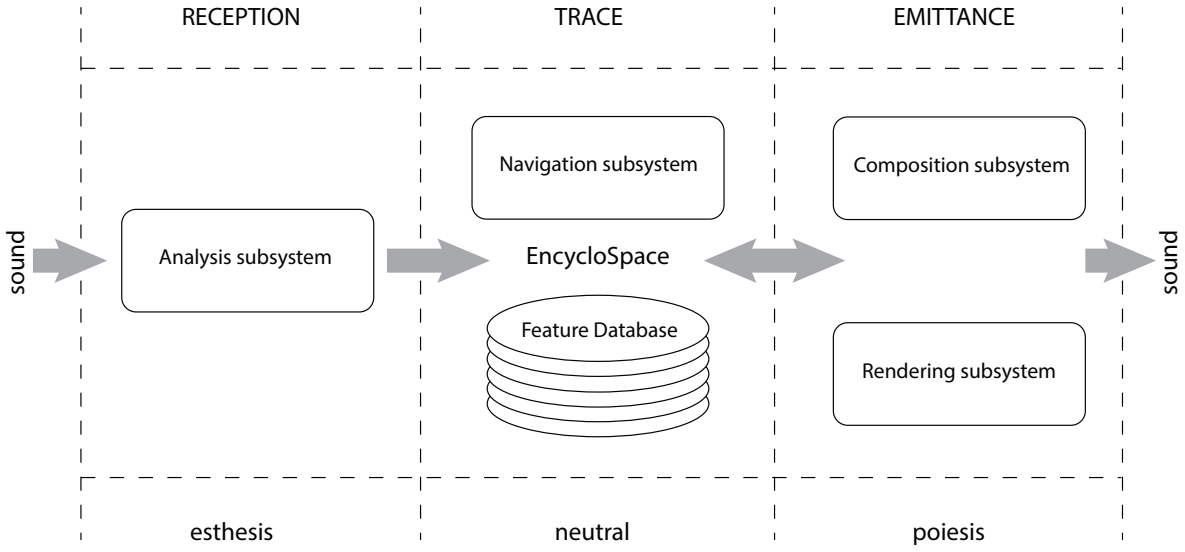


Figure 2.2: System communication schema

2.2.2.1 Productive function (Emittance)

The emittance layer of the system corresponds to its primary function of producing the musical result. This function is carried out by two functional parts working in concert:

- Composition subsystem
- Rendering subsystem

The composition subsystem comprises tools to create compositional structures on the macro and micro time scales. It also resolves logic of temporal aspects of composition. Its output consists of compositional objects which collectively contain parametric definitions of the composition. This structure is then fed into the rendering subsystem which parses parametric structures, resolves probabilistic constructs into deterministic sequences, converts logical time into physical time, maps sound data onto physical timeline and renders the result. Composition and rendering subsystems are described in detail in sections 2.5.2, 3.4 and 2.5.3, 3.5 correspondingly.

2.2.2.2 Receptive function (Reception)

The reception layer represents the capacity of a system to evaluate entities. In the context of this project, this can be specified as the ability to parameterize incoming sounds via process of feature extraction. While the presence of the emittance function is shared by all compositional software, the presence of the receptive function is not. However in our project, both emittance and reception are inexorably intertwined in order to achieve compositional goals. The reception layer is implemented as the *analysis subsystem*. Implementation details are given in sections 2.4 and 3.3.

2.2.2.3 Trace function (Transport)

The transport layer is the glue of the system, operating as a bridge between productive and receptive layers. Its physical organization provides a persistence means for the EncycloSpace of the system. The receptive layer augments EncycloSpace by creating new entries out of freshly segmented elements and newly analyzed features. The productive layer uses the *navigation subsystem* to cruise EncycloSpace in search of compositional data and also to create new entries from recombining elements.
Further sections discuss design considerations with respect to each functional layer.

2.3 Transport layer: Media database

Media database considerations will address:

- Database general architecture
- Structure of database relations
- Building-block elements

2.3.1 General architecture

The media database represents the system's body of knowledge, EncycloSpace, memory. Structure, robustness and interface of the database are instrumental for the functioning of both analytic and compositional phases of work with the system. The requirements for media database come in a variety of aspects:

- Quantitative aspect: ability to store arbitrarily ample quantities of data.
- Speed: expedience in data creation, update, search and retrieval.
- Integrity: ability to protect data from ending up in inconsistent states.
- Flexibility: accommodating data of complex structures.
- Concurrency: allowing multiple simultaneous contexts of data access.

- Compression: ability to conserve storage space by applying data compression.
- Free access: database systems that offer free tools for non-commercial users.
- Availability of Python tools: this requirement is included due to our choice of the development platform.

Different aspects may suggest different preferences in the choice of a database management system. For instance, HDF[2] platform [53] gained popularity as storage model for scientific data due to its data structure flexibility, access speed and compression. It is accepted in some music research and information retrieval systems such as [75] and [19]. However, HDF, which is based on file access, does not respond well to requirements of concurrency and data integrity. RDBMS[3] systems are on the opposite side — they support integrity, concurrency and distributed access to data but they experience some processing overhead and require special tricks for handling hierarchical data. Object-oriented DBMS could remedy the latter. Some specialized media database approaches were also proposed usually with the goal of optimizing some of the aspects listed above. For instance LSH[4] format described in [106] drastically reduces time of search by similarity in extra-large databases. This method was employed in [31] and generally can be welcomed in any realtime concatenative synthesis application.

Since no single DBMS solution seems to respond equally well to all of the media database requirements the choice has to be informed by the priorities of the DBMS client system. Settling on corpus data structures suggested a database architecture approach based on ORM[5] layer [17] running over PostgreSQL RDBMS [54].

[2]Hierarchical Data Format
[3]Relational Database Management System
[4]Locality-Sensitive Hashing
[5]Object-Relational Mapping

2.3.2 Structure of database relations

Providing material connection between poiesis and esthesis of the system[6], the media database needs proper planning of its data structures. This concerns specifications of the database schema — a set of interconnected relations that define the structure of the system's EncycloSpace. Since the latter integrates circulation of data in both directions of emittance and reception it is instrumental that the database schema fully supports semiosis as defined in the ontological model (1.1.3.3). In other words, the database schema should allow elements to refer to other elements when defining their content thus supporting element's appearance on either side of signification act, i.e. *signifier* and *signified*.

A database schema for this project was devised after studying related work in other corpus-based systems. The reception part is largely designed following the model described in [100], with a number of modifications and extensions. The emittance part of the database schema was designed afresh. Its structure facilitates manipulations of sound elements on the basis of extracted semantics in order to foster their compositional recombinations. Database schema details follow further in 3.2.

2.3.3 Building-block element

The building-block element is a data structure that serves as a functional template which holds descriptions for any sound entity contained in the media database. This structure defines properties of sound in terms of its physical storage location (typically — sound file) and temporal boundaries within the file. Building-block element (or simply *element*) is also a structural unit to use for describing sound as ordered aggregate of smaller elements. This allows the creation of structural hierarchies of sound elements which also reflects how sounds are parsed by human perception and cognition mechanisms[43].

[6]Both analysis and composition subsystems use the database to carry out their function

2.4 Receptive layer: Analysis subsystem

From the perspective of communicatiion with EncycloSpace the *analysis subsystem* carries out one of the most important functions of the whole system - converting real-world phenomena into knowledge. Real-world phenomena in this case includes recorded sound, knowledge — features extracted from it and categories assigned to it. By means of multi-dimensional analysis the system materializes an esthesis line of ontological communication. Its primary function is not to create a musical result by itself but rather to build an informational background that can be used to create the musical result. Through this connection, the analytic subsystem influences the potentiality of the system's poiesis.

From a practical perspective sound analysis is a multiple-step process:

1. Audio segmentation
2. Feature extraction
3. Categorical description

the first two steps are executed in the given order — first segmental boundaries are figured out and then feature analysis is executed with respect to material contained in each segment. Category assignment can be done before and/or after segmentation/feature extraction due to its ability to translate class membership along lines of structural inheritance (see details in 2.4.3).

2.4.1 Audio segmentation

As stated in [11]:

> An important step towards a musically useful parametrization is the segmentation of a sound into regions that are homogenous in terms of a set of sound attributes. The goal is to identify regions that, using the signal properties, can then be classified in terms of their content. This way

> we can identify and extract region attributes that will give higher-level control over the sound.

Critical role of segmentation is also corroborated in [110]:

> Segmentation alone can determine the quality of a corpus of sounds. Improper segmentation can lead to a loss of information and sub-par search results.

and in [25]:

> Temporal segmentation of an audio stream into shorter elements is a fundamental step in the transformation of sounds into semantic objects.

Depending on application segmentation can significantly differ in terms of temporal range within which segment boundaries are sought. For instance structural segmentation, assisting in genre classification, would look for temporal boundaries on a scale of musical sections, while melody analyzing segmentation will be more interested in finding segments on a timescale of a single music event. Other segmentation contexts can include: segmentation for the purpose of music practicing [123], field recordings catalogization [71], temporal separation of speech and music in radio signals [49] and more.

In the context of this project, analysis is applied to continuous regions of sound in order to increase the volume of an EncycloSpace of the system. Each analyzed region is assigned an id and becomes a navigational point of entrance within the EncycloSpace. Methods of audio segmentation have a direct impact on the timescale structure of the resulting searchable space.

Even though segmentation belongs to analytical processes, i.e. it is a component of esthesis, it is also a part of the compositional strategy (poiesis). This is because it defines navigable space for the productive phase which, in turn, can be invested in very different time scales for search of composition elements. In general corpus-based

composition system should support all compositionally meaningful time scales. Audio segmentation can be approached in four general modes:

- Manual segmentation
- Unsupervised segmentation
- Supervised segmentation
- Mixture of the above

2.4.1.1 Manual segmentation

In a manual mode, segment boundaries are obtained solely as a result of actions administered by a human operator. In this mode, the software's role is limited to manual editing and visualization of segments. While hardly useful for musicological tasks (except for specifying the ground truth[7]) the manual mode still can be helpful for composition-grade segmentation where precision is critical. The serious downside of this method is its impracticality in situations where hundreds or thousands of segments have to be specified.

2.4.1.2 Unsupervised (automatic) segmentation

The segmentation algorithm goes without any guidance from a human expert (or any other external source) and only needs sound input to generate segment boundaries. Segmentation criteria can be chosen from wide variety of features and applied in different combinations. Some approaches to unsupervised segmentation are discussed in [92] and [15].

[7]Ground truth is a term used to designate absolutely correct solutions. It is omnipresent in machine learning applications as a tool to evaluate validity of algorithmic output.

2.4.1.3 Supervised segmentation

To draw conclusions about segment boundaries supervised algorithms use pre-existing knowledge. This group of methods is at home in the contexts where specific expectations exist about the characteristics of a sound input. Supervised segmentation works best when the score accompanies the recording that is a subject of segmentation or when the category space of intended classification is known beforehand. For an omni-source compositional system this is unlikely the case — its knowledge base should accommodate sounds brought in from arbitrary sources and of discretionary spectro-morphological characteristics. Due to this openness and the significant indeterminacy of taxonomy space in the compositional process, supervised methods might have limited applicability.

2.4.1.4 Mixed-mode segmentation

Segmentation modes can be mixed to improve the correctness and effectiveness of analysis.

2.4.2 Feature analysis

Whereas segmentation is critical for temporal structuring of material, audio-content based feature analysis is responsible for the parametric variety of its representation inside EncycloSpace. Each feature represents a single dimension in multi-dimensional semantic space. More analytic dimensions mean more possibilities in compositional decision-making based on database navigation. Too many dimensions, however, can create problems with processing time during proximity-based searches where all dimensions partake in computing the feature distance between sound elements. Therefore, the set of analytic descriptors should be chosen amply but wisely in order to represent the most compositionally salient characteristics while avoiding unnecessary

duplications or semantics of marginal importance. Literature on music informatics has a broad range of publications directly or implicitly discussing sets of sound descriptors for musicological tasks. Examples of such discussions can be found for instance in [89], [11], [100]. Part 4 of MPEG-7 standard also contains definitions of descriptors related to sound representations [9].

2.4.2.1 Description level

On the scale of description level feature descriptors are frequently divided into three groups:

- Low-level descriptors
- Mid-level descriptors
- High-level descirptors

Low-level descriptors (LLDs) (also called instantaneous descriptors) are computed across short-time window typically on a scale of up to one hundred milliseconds. This information averaged over an analysis frame is attributed to a specific time point usually corresponding to the middle of the frame. Examples of LLDs that can be extracted from time domain representation include energy, zero-crossing rate, autocorrelation; examples of frequency domain LLDs — spectral moments (centroid, spread, skewness and kurtosis), spectral energy, spectral roll-off etc.

Mid-level descriptors (MLDs) result from processing of low-level descriptors normally without aggregation of LLDs over time. MFCCs[8], BSCBs'[9] energy, spectral peaks, harmonic spectrum are examples of Mid-level descriptors.

High-level descriptors (HLDs) result from aggregation over time of low-level and mid-level descriptors applied in various ways and combinations. They compute

[8]Mel-Frequency Cepstral Coefficients
[9]Bark Scale Critical Bands

characteristics more intuitively understood than LLDs and MLDs, such as melody, rhythm, key and harmony.

2.4.2.2 Temporal variability

Sound is a time-varying phenomenon and so are some of its descriptors (especially LLDs and MLDs). By temporal variability features can be:

- Static
- Dynamic

Static features describe sound as a whole using a single value. Examples of static features are attack time, temporal mean, temporal density, onset cardinality etc.

Dynamic features depict changes of a given sound semantics over its lifetime and result in a time series of values. This poses certain challenge to further usage of these data. How much detail should we store in database and in what form? Keeping all values will make the data unusable in a compositional context because there will be no simple way to interpret their meaning. At the opposite extreme we could simply store an average of the series, but that would be too reductive — no temporal variability will be preserved in that case. A compromise can be found in a middle-ground solution: to apply certain statistical post-processing to the time series. This will reduce the amount of data to just a few scalars, but will sufficiently preserve characterization of the descriptor's temporal behavior. This approach is discussed in detail in [100].

It is worth noting that not all time-varying features can be effectively processed that way. As an example let's consider a sound fragment where different pitches emerge at different times for different durations. The aforementioned processing will treat pitches as a statistical mass and will not preserve their vertical relations which can

be very useful in compositional context. With that sort of feature, we need to apply strategies that don't destroy their compositional merits.

2.4.2.3 Physical and psychometric

Sound is known as a double-faced phenomenon: on the one hand it has physical properties such as frequency and intensity that can be measured objectively, and on the other it has psychometric characteristics, such as pitch and loudness, that can only emerge phenomenologically. In this example, two pairs are causally related but have different scales of representation — linear growth in psychometric characteristics corresponds to exponential growth in physical ones. This creates additional considerations for choosing the proper feature scale. Designing a system for compositional purposes dictates a preference for psychometric units as they are more closely related to the listening experience. This preference is instantiated in the choice of analysis descriptors presented further (see 3.3.1).

2.4.3 Categorical description

Whereas feature analysis unfolds quantitative characteristics of sound, categorical description characterizes sounds qualitatively in the form of class membership. One sound can belong to different category classes, each of which outlines different aspects of its production: source, excitation material, resonator material, participating states of matter, production techniques, location, spectro-morphological characteristics etc. Due to its operational simplicity class membership is a convenient method of sounds segregation and category space is a natural part of music EncycloSpace. To build the sound classification system we have to answer two general questions:

1. What is the taxonomy of class categories for sound?
2. What are the methods of classification?

2.4.3.1 Class taxonomy

As stated in [76]:

> Sound classification is one of the most complex unsolved problems in musicology. The reasons are triple: First, the communicative determinants of sounds are not clear. Second, sound varieties are ... completely unclassified objects. Third, the semantic charge of sounds is a substantial constraint for classification; there is no good classification without semantic constraints.

In view of the aforementioned problems, existing sound classification taxonomies vary significantly depending on the context in which they are utilized. Typical contexts in music informatics include genre classification and instrument classification. For an omni-source compositional context, we are interested in the most general approach capable of embracing arbitrary sound morphologies.

An initiative called Semantic Web features Music Ontology — a formal framework that defines protocol called RDF[10] for creating music taxonomies by the use of machine readable form[96]. The framework offers cross-class dereferencing and can be expanded by any registered party. One of the framework's goals is to create a platform for the construction of music databases and information exchange. At the time of this writing, there are 54 registered taxonomy classes[8]. Unfortunately, none of them covers sound sources except for the class Instruments. For our project's purposes this scope is too narrow. Moreover, spectro-morphological sound characteristics are also not present among Music Ontology classes.

However helpful a reference to standardized general sound taxonomy could be, its absence is not necessarily critical for this project — given the appropriate tools, a user of the system should be able to create sound taxonomies of her own choice.

[10]Resource Description Framework

2.4.3.2 Classification methods

Once the category taxonomy is established, the category space can be filled up by applying sound classification methods. As in the case of segmentation classification, this can be done with different degrees of automatization:

- Manual classification
- Automatic classification
- Hybrid classification

The manual classification method is self-descriptive — all category assignment is done by a human expert. Considering that in omni-source environment a sound taxonomy is likely to be unbound, we should expect much of category assignment be done manually. For this scenario appropriate computer tools should be designed to make manual assignment efficient.

Automatic classification is based on applying machine learning methods autonomously. Classification mainly uses descriptors derived from audio but it can also make inferences by parsing textual information obtained from file names and file metadata. Some strategies of automatic classification are discussed in [91] and [70]. Regardless of the strategy automatic classifiers operate within predefined taxonomies. Environments with unbound taxonomies (like ours) are likely to create certain systemic problems for automatic classification.

Hybrid classification combines automatic and manual methods. This can be useful in environments where automatic methods can cover only part of taxonomy. For instance, in omni-source environment automatic classification could cover instrumental sources while the rest could be done manually.

2.4.3.3 Ontological binding

Sound classification can be done using either objective or subjective categories. The former group describes sound *as produced*, the latter — *as perceived*. Examples of objective taxonomies: instrument, object, material, playing technique; examples of subjective ones: emotion, character, metaphor. Characterizing sound as *violin pizzicato* or *yarn mallet* uses objective classifiers, referring to it as *creepy, buzzy* or *uplifting* uses a subjective taxonomy.

In certain circumstances an objective category can be used in place of a subjective one. For instance, if the sound of a cat reminds one of a baby crying — should it be put in the database as *a cat* or as *a baby*? Certainly, we should keep both characterizations with the distinction that the sound *as produced* is of a cat and *as perceived* is of a baby. To make such a split classification the assigned categories should be accompanied by flags with values showing whether the characterization is real (objective) or imaginary (subjective). Such a method would allow not only the search for sounds which *are something* but also for sounds which *sound like something or remind one of something.* There can be plenty of such cases especially in the presence of abstract sounds, synthesized or otherwise produced. Because human perception always attempts to resolve the provenance of acoustic phenomena[107], it invariably takes sound as "evidence to distinct environmental causes"[23]. Allowing categories to have *imaginary* status provides a practical solution to this.

Along with the qualitative categorization for abstract sounds, we can also employ spectro-morphological schemas such as Schaeffers *Mass-Facture* topology[81]. However, this aspect is left to the discretion of user.

2.5 Productive layer: Navigation, Composition, Rendering

The productive layer comprises subsystems whose functions are combined in the production of the musical result. Such subsystems are:

- Database navigation subsystem
- Composition subsystem
- Performance subsystem

2.5.1 Database navigation

Obtaining material from the media database during the compositional process is an essential part of the functionality within corpus-based systems. The main aspects to consider are:

- Flexibility of acquisition,
- Receptive/Productive access to EncycloSpace.

2.5.1.1 Flexibility of data acquisition

Access to EncycleSpace needs to be diversified as much as possible, i.e. database navigation needs to be equipped with a flexible interface allowing a full search range. Search contexts should be dynamically configurable allowing for vari-dimensional queries. Large databases can present problems with scalability. In [105] algorithms are proposed which address that problem by combining kD-trees and mass-spring models for fast similarity searches in high-dimensional databases. The SQL syntax traditionally lacks convenient clause constructs for proximity searches (this problem is discussed in [14]). To amend that problem, on top of the SQL we utilize an object-relational mapping layer.

2.5.1.2 Receptive/Productive access

Data sets, obtained via navigation (receptive access), should be able to obtain a persistent state within EncycloSpace via accretion (productive access). Thus obtained, these collections exemplify the first stage of data recombinance. They can be directly mapped to compositional objects. Receptive/Productive navigation is one of the pivotal points in the system's functionality.

2.5.2 Composition subsystem

The poiesis component is immanent to all computer-aided compositional systems. Actual functionality differs in scope and structure depending on the software's system identification (2.1.1).

Inherent reliance on sound analysis and corpus makes this software's compositional subsystem share some functionality with other corpus-based compositional systems. Most of these convey either *concatenative synthesis* or *music mosaicing* paradigms. There is no standard or manifesto of how these terms are defined, but for the purpose of this discussion we define them as follows:

- Music mosaicing — requires the so-called *target sound* to serve as a criterion for database elements selection and as an organizational structure for their assembly on the timeline. So, strategically the decision-making in *music mosaicing* is considerably deterministic.
- Concatenative synthesis — allows interactive navigation through EncycloSpace, which is usually presented as a two-dimensional field where each dimension can represent one of sound descriptors stored in the database. This approach allows improvisatory work with EncycloSpace but seems to lack in algorithmic capacity.

Both methods use only *receptive* mode of navigation.

This project aims to steer away from these conceptual restraints and arrive at a more general solution with full ontological access to EncycloSpace. Overcoming compositional limitations of traditional concatenative synthesis and music mosaicing are addressed, for instance, in [110] where the proposition is to combine navigational strategies of both idioms to attain additional degrees of freedom in material selection. Other useful techniques are suggested, such as alternative segmental projections for sound units or dynamic unit definitions to promote hierarchical relationships and mutability of database elements.

Another approach derived from the formalisms of a theory of *sound-types* is introduced in [33] and further elaborated in [34]. The proposed model suggests a set of methods towards sound hybridizations based on specific reduction and ordering techniques. Reduction is done by signal automization, low-level feature computation and subsequent clustering of sound atoms in order to infer sound-types — aggregated representations of sound atoms. Along with sound-types, probability matrices are computed that describe likelihoods of transitions between sound types thus capturing temporal relationships on specified degrees of order. Sound synthesis is then done on the basis of querying sound-types dictionary and applying transitional rules to yield sound result. The dictionary and the rules function as the EncycloSpace for the production phase. The overall functionality of the sound-types model, which is a combination of audio compression and probabilistic production, somewhat deviates from strictly compositional focus but it contains compelling ideas that could be explored. From a compositional perspective perhaps the most problematic feature of the model is that, as the sound database grows, its abstraction degree increases as well, which diminishes the model's relative diversity of sound representation.

The *AudioGuide* project presented in [56] bears a distinct compositional perspective in approach to corpus-based synthesis. One of its main premises involves treating

sound as a source for extracting gestural trajectories and manipulating these in order to derive degrees of behavioral variance. Features extracted from the target are not necessarily immutable — they can be manipulated to warp database data selection in order to derive new correlated gestural contours. Furthermore, during segment selection in the presence of target sound, the AudioGuide considers different methods of feature normalization to further negotiate between direct imitation and metaphoric transcription. Applying weights to feature selection is another way to influence matching process in favor of variability. Resulting solution space can become high-dimensional which creates problems for its navigation. AudioGuide uses dimensionality reduction methods, described in [103], based on *multi-dimensional scaling* which allows to clusterize the solution space. One of the most notable features in AudioGuide is a method of *subtractive selection* which matches target in multiple steps: in each step residual spectrum is computed which is equal to spectral mismatch between target and selected database segment; this residual spectrum becomes a new target and the search continues until residual spectrum does not contain significant energy anymore. Then thus obtained set of segments is applied in concert to replace the target with minimal error. The subtractive selection's principle of incremental vertical superposition of segments allows to work towards better precision in the synthesis process and use corpus with the feature space not necessarily overlapping with that of a target's.

2.5.2.1 Structure of Composition

Systems described above predominantly focus research on micro time-scales, i.e. from the level of sample to gesture or phrase. Larger compositional structures, those of macro time scales, are mostly left to one of two options: either free-form interactive browsing of database (concatenative synthesis) or replicated from target sound (audio

mosaicing). In both scenarios macro structure emerges from prolonged execution of a single process in a manner of bottom-up composing (as defined in [82]).

This project is driven by the motivation to create a full time-scale solution for corpus-based composition. Along with researching micro-scale methods of sound synthesis, this project also promotes the availability of the macro-scale constructs, which allows this project to add top-down composing to the already available bottom-up counterpart.

The overall structure of the compositional subsystem is a four-tier layout shown in figure 2.3.

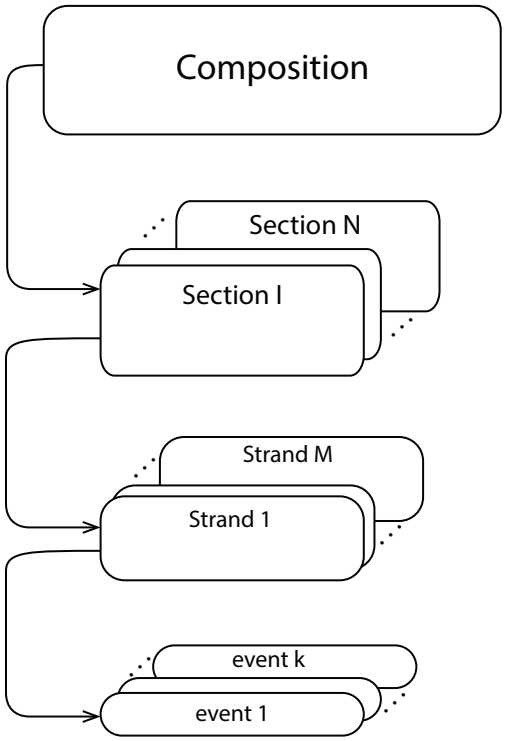


Figure 2.3: Structural layout of composition

Composition is a top-level structure that represents the whole piece. It consists of sections which are deployed in succession during rendering.

Section represents a large-scale division of music similar to that of a movement. Each section consists of strands which are deployed in parallel during rendering.

Strand is a construct similar to a single voice in instrumental music but this analogy is rather relaxed. A strand consists of events which are deployed in succession during rendering.

Event is the smallest unit, but only in the structural sense. Its duration is unrestricted and is subject to compositional decision-making. A structural event, as presented here, is not equal to a sound object — these are completely different entities. In the simplest case, an event can contain a single sound element but most of the time it will contain collections of sound elements. Details about relationships between events and sound elements are provided in 3.4.5.4.

2.5.2.2 Composing with EncycloSpace, or *NOTA*-transform

To better describe and compare corpus-based strategies we define terminology based on the operational pattern called *NOTA*-transform. It refers to general principles of the compositional algorithmic process defined on EncycloSpace: *NOTA*-transform is applied to the latter thus producing a new musical object (figure 2.4).

Figure 2.4: Transforming EncycloSpace into musical object

NOTA-transform designates four transformational phases disabbreviated as **N**avigation, **O**rdering, **T**emporalization and **A**daptation. The phases are chained into a decision-making pipeline with each phase's source domain being either the whole EncycloSpace or a projection thereof. Transformation process goes through

producing successive projections until it reaches a renderable state in which it can be utilized as musical object.

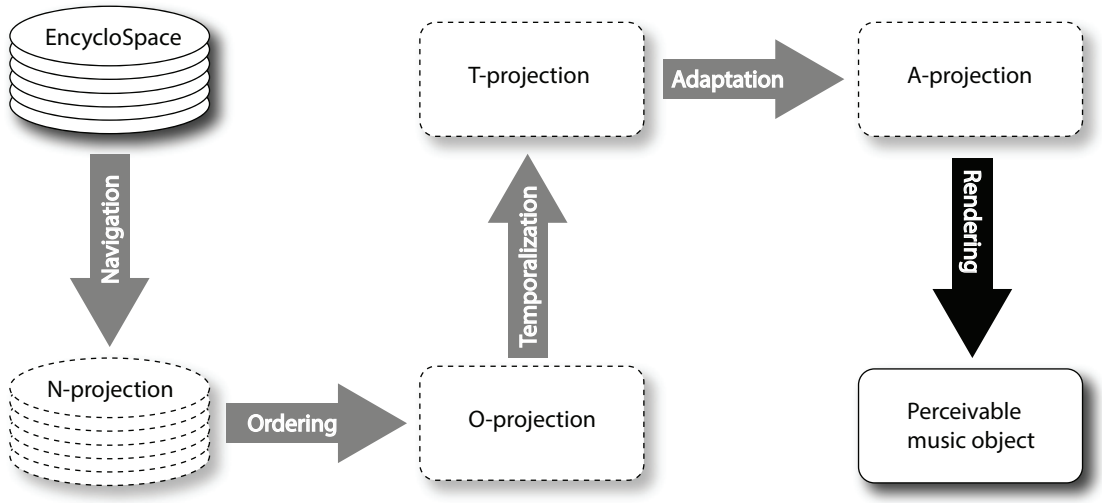


Figure 2.5: $NOTA$-transform phases

$NOTA$-transform can be concisely defined as a four-step formalism. Let's first define EncycloSpace of length L as

$$E = \{\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_l\} \tag{2.1}$$

where ε_i is element as defined in 2.3.3.

Step 1. Navigation transformation. Given an EncycloSpace E it is possible to define **N-projection** as a result of *navigation transformation* applied to E:

$$E \xrightarrow{\nu(C_N)} P_N \tag{2.2}$$

where ν is *navigation transformation* and C_N — context which defines a set of constraints for navigation. N-projection is a E's subset of N elements selected with compliance to c:

$$P_N = \{\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n\} \tag{2.3}$$

It should be noted that both E and P_N are sets, i.e. there is neither ordering nor duplication of elements.

Step 2. Ordering transformation. Given N-projection P_N, O-projection P_O is defined as an outcome of *ordering transformation* applied to P_N:

$$P_N \xrightarrow{\omega(C_O)} P_O \tag{2.4}$$

where ω is *ordering transformation* and C_O — context which defines how ordering should be applied to P_N. O-projection is based on members of P_N but it is not a *set*, it is a *sequence*, i.e. duplication of elements is allowed and they are arranged in an order of precedence with total length of r:

$$P_O = \{\varepsilon_{1,i(1)}, \varepsilon_{2,i(2)}, \ldots, \varepsilon_{r,i(r)}\} \tag{2.5}$$

Elements indexing now has two parts: indices of the first half — $1, 2, \ldots, r$ — define sequential ordering and indices of the second half $i(1), i(2), \ldots, i(r)$ refer to the original index of an element in P_N.

Step 3. Temporalization transformation. Given O-projection P_O, T-projection P_T can be defined as a result of *temporalization transformation* applied to P_O:

$$P_O \xrightarrow{\tau(C_T)} P_T \tag{2.6}$$

where τ is *temporalization transformation* and C_T — context which defines how temporal position and duration should be assigned to the elements of P_O. The result is the sequence of the same length as P_O

$$P_T = \{\varepsilon^{t_1,d_1}_{1,i(1)}, \varepsilon^{t_2,d_2}_{2,i(2)}, \ldots, \varepsilon^{t_r,d_r}_{r,i(r)}\} \tag{2.7}$$

where lower indices are the same as in P_O, upper indices $t_i \in \{t_1, t_2, \ldots, t_r\}$ — computed time locations and upper indices $d_i \in \{d_1, d_2, \ldots, d_r\}$ — computed durations[11].

Step 4. Adaptation transformation. And finally, with T-projection P_T as input, A-projection P_A is defined as a result of *adaptation transformation* applied to P_T:

$$P_T \xrightarrow{\alpha(C_A)} P_A \tag{2.8}$$

where α is *adaptation transformation* and C_A — adaptation context which specifies parameters of the signal processing to be applied to the elements of P_T. The result is the sequence of the same length and time positions as P_T.

$$P_A = \{\varepsilon^{t_1,d_1}_{1,i(1)}, \varepsilon^{t_2,d_2}_{2,i(2)}, \ldots, \varepsilon^{t_r,d_r}_{r,i(r)}\} \tag{2.9}$$

The adaptation phase takes care of the sound element's final adjustments necessary to produce the desirable aural result.

Each application of *NOTA*-transform creates processing pipeline $E \to P_N \to P_O \to P_T \to P_A$ which converts EncycloSpace into a renderable music object P_A of arbitrary timescale — from a single note-like event to the whole composition. The exact result depends on four structures:

- Navigation context C_N (2.2)
- Ordering context C_O (2.4)
- Temporal context C_T (2.6)

[11]Computed duration should not be confused with element's natural duration produced by segmentation. Computed durations result from *temporalization transformation* so they may or may not coincide with the elements' natural durations. In the latter case concatenation techniques should be applied to resolve mismatch between natural and computed durations

- Adaptation context C_A (2.8)

2.5.2.3 Navigation context

Navigation context is defined as a set of constraints applied to the process of elements selection. It can be constructed as an SQL query combining series of SELECT and JOIN clauses predicated upon elements features. Execution of such a query will return a subspace of elements whose features satisfy the constraints.

2.5.2.4 Ordering context

Navigation transformation returns an unordered selection of elements. A decision has to be made about the number and succession of elements for the actual use in the composition. Ordering context is defined as an algorithmic construct which solves this task by computing a sequence of elements indices which define their number and order. Indices are allowed to duplicate which means that elements can participate in the sequence an arbitrary number of times. More details on ordering contexts can be found in 3.4.2.

2.5.2.5 Temporal context

In composition, each sound object has its time position and duration. Ordered sequence defines a progression of elements which certainly can be used verbatim for compositional deployment. In such a case, each element would use its natural duration and its time position would be provided by the preceding element's end time. This scenario, however, leaves out the temporal plan which the composer might have and which may arrange elements very differently in the contexts of temporal initiation and duration. Temporal context is a construct dedicated to providing algorithmic temporal control of compositional objects. It consists of *time structures*, each one describing specific formation of time continuum in terms of divisions and propor-

tions. Temporal context is a part of *section* that uses it to apply temporal control to strands. Usage of the temporal context is not mandatory, as strands can be perfectly rendered without it. However, it is a convenient tool for rhythmic (or polyrhythmic) organization of sound elements.

Time structure consists of symbolic units describing how temporal continuum should be structured in terms of logical time points. Each strand can be bound to exactly one *time structure* which should be selected from the temporal context of its parent section. Strand can use time structure to align its events against specific types of time points during rendering. Mapping of time structures to strands is flexible - a single time structure can be assigned to all strands and equally each strand can be assigned an individual time structure.

2.5.2.6 Adaptation context

To produce quality sound result, elements of T-projection may require an additional processing concerned with possible concatenation issues[12] or final morphological adjustments. The *adaptation context* describes the parameters of those adjustemnts.

2.5.2.7 Indeterminacy

The creation and usage of compositional structures can be completely deterministic but it does not have to be. Four transformational contexts of *NOTA*-transform, C_N, C_O C_T and C_A, epitomize decision-making nodes which can be constructed with variable degrees of indeterminacy. Compositional events can be created in partially abstracted state with respect to their content, elements order, their temporal position and/or duration. The higher indeterminacy the closer compositional process to that

[12]Since overlapping neighbor elements of the T-projection may come from different sources, there can be a perceivable discontinuity that impacts the quality of a sound result. This discontinuity is called *concatenation cost* [100]

of *meta-composition* [109]. All compositional elements which were instantiated in probabilistic space will be resolved during the initial phase of rendering.

2.5.2.8 Parametric automation

Strands and events can be controlled by assignable time functions. This allows fine-grain control over their parameters during rendering. To add more layers of indeterminacy to the decision-making process automation functions can also be stochastic.

2.5.3 Rendering subsystem

Rendering process dereferences compositional symbolisms and converts them into physical result. The result can be a waveform and/or symbolic notation. Ideally the system should be able to assign preferred rendering mode to different parts of composition thus allowing to mix acousmatic and instrumental writing.

The system design considerations discussed in this chapter were used as guidelines to approach an implementation of the software presented in the next chapter.

Chapter 3

sEl: An implementation of recombinant corpus-based omni-source compositional software

This chapter describes *sEl*[1] - a CAAC framework that implements concepts described in the previous chapter. sEl is an open-source software written in the Python programming language. It is implemented as an integrated system that incorporates a number of third-party open-source components as a foundation to be used for adding sEl's own functionality. The following sections describe typical workflow and implementation details for each of the functional subsystems.

3.1 General workflow

The software has two core areas, analytic and compositional, the functionality of which relies on sound material stored in a system database. Populating the database with sound data is the earliest stage of the workflow.

[1] sEl stands for *sound element*

3.1.1 Audio pool

To organize sound files sEl uses a repository called *audio pool* which is simply a dedicated file system location, either local or networked. All sound file paths stored in the database are represented as relative with respect to the audio pool's root. This makes sound locations stored in the database independent from the location of the audio pool itself.

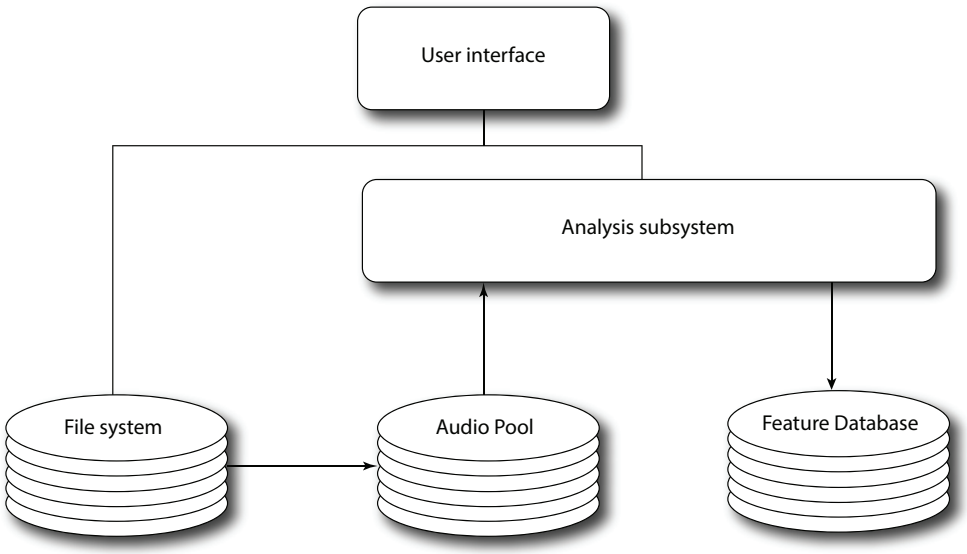


Figure 3.1: Database workflow

The audio pool can be populated with sounds either by using the sEl user interface or by utilizing regular operating system tools. sEl *audio pool* interface is shown in figure 3.2: the file system browser[2] in the lower half is used to drag sounds to an audio pool editor shown in the upper half. If the sound file is accompanied by a segmentation file of a known format, that file will be automatically transferred to the audio pool. The audio pool editor displays information about the pool's structure and containment. This information is presented as per-folder file counters and binary flags marking the files' registration with the database.

[2]File system browser shows only audio files

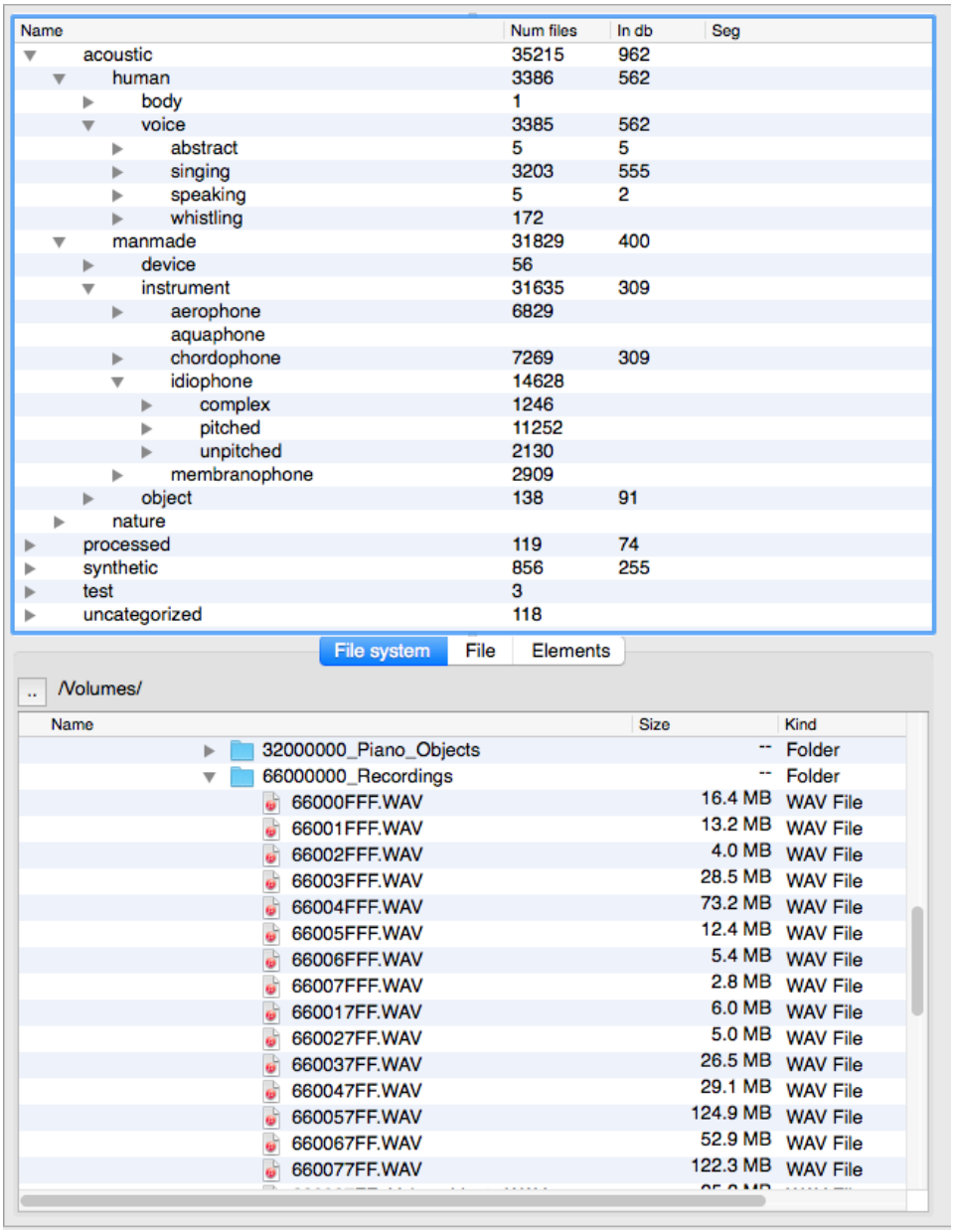


Figure 3.2: Audio pool user interface

3.1.2 Registering sounds with database

Files stored in an audio pool can be registered with the database by using context menu commands shown in Fig.3.3. For each file registered with the database, a corresponding database element is created which represents the whole file. Such an element is called a *root element*.

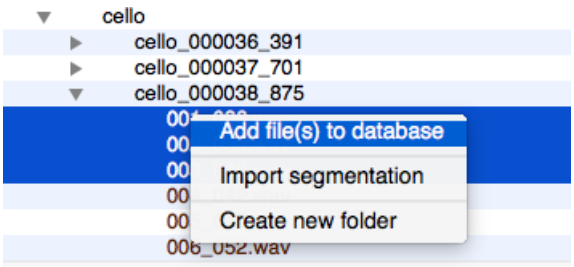


Figure 3.3: Audio pool context menu commands

3.1.2.1 Automatic segmentation import

If a registered sound file is accompanied by a segmentation file or contains segments internally as metadata, the segmentation data is automatically used to create database elements, one element per segment[3]. The segments are then further analyzed to infer containment hierarchies and sequential ordering. These structural relationships are also written to the database.

3.1.2.2 Automatic categorization

During the sound file registration an automatic categorization is attempted based on textual analysis of the file's path, name, internal metadata or the presence of a categorization file prepared by an external utility.

3.1.2.3 Sound file overview

Detailed information about registered sound files can be examined in the *File information viewer* shown in figure 3.4.

3.1.3 Assigning categories

During database registration categories can be automatically inferred and assigned to sound files. In addition to automatic categorization more categories can be assigned

[3]Currently the following formats are recognized: SonicVisualizer XML export and Adobe Audition region markers metadata.

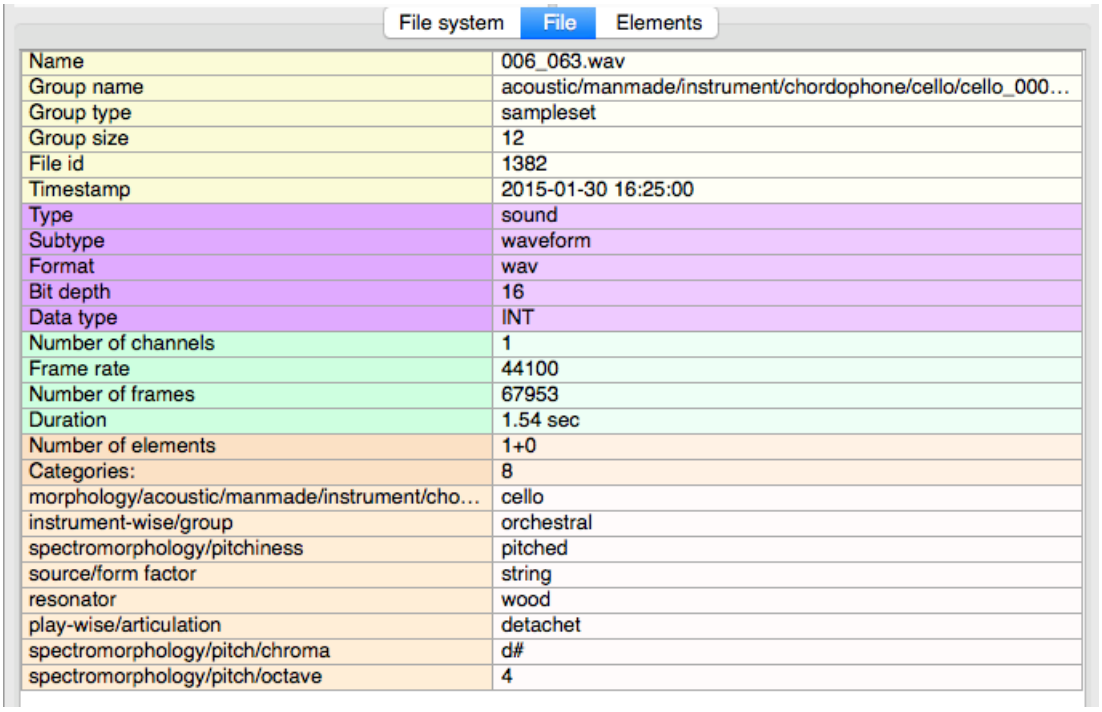


Figure 3.4: File information viewer

manually in the *Category editor* shown in figure 3.5. A taxonomy shown in the figure comes with the software and defines six top category classes:

- *Morphology* class — represents nomenclature of sound sources as physical objects.
- *Source* class — taxonomy of sound excitation component's form factor and state of matter.
- *Resonator* class — taxonomy of resonating material.
- *Instrument-wise* class — a taxonomy applicable if sound source is a musical instrument.
- *Play-wise* class — taxonomy representing performative aspects of sound production.
- *Spectro-morphology* — taxonomy of perception related characteristics of sound.

The default taxonomy is presented in detail in Appendix B.

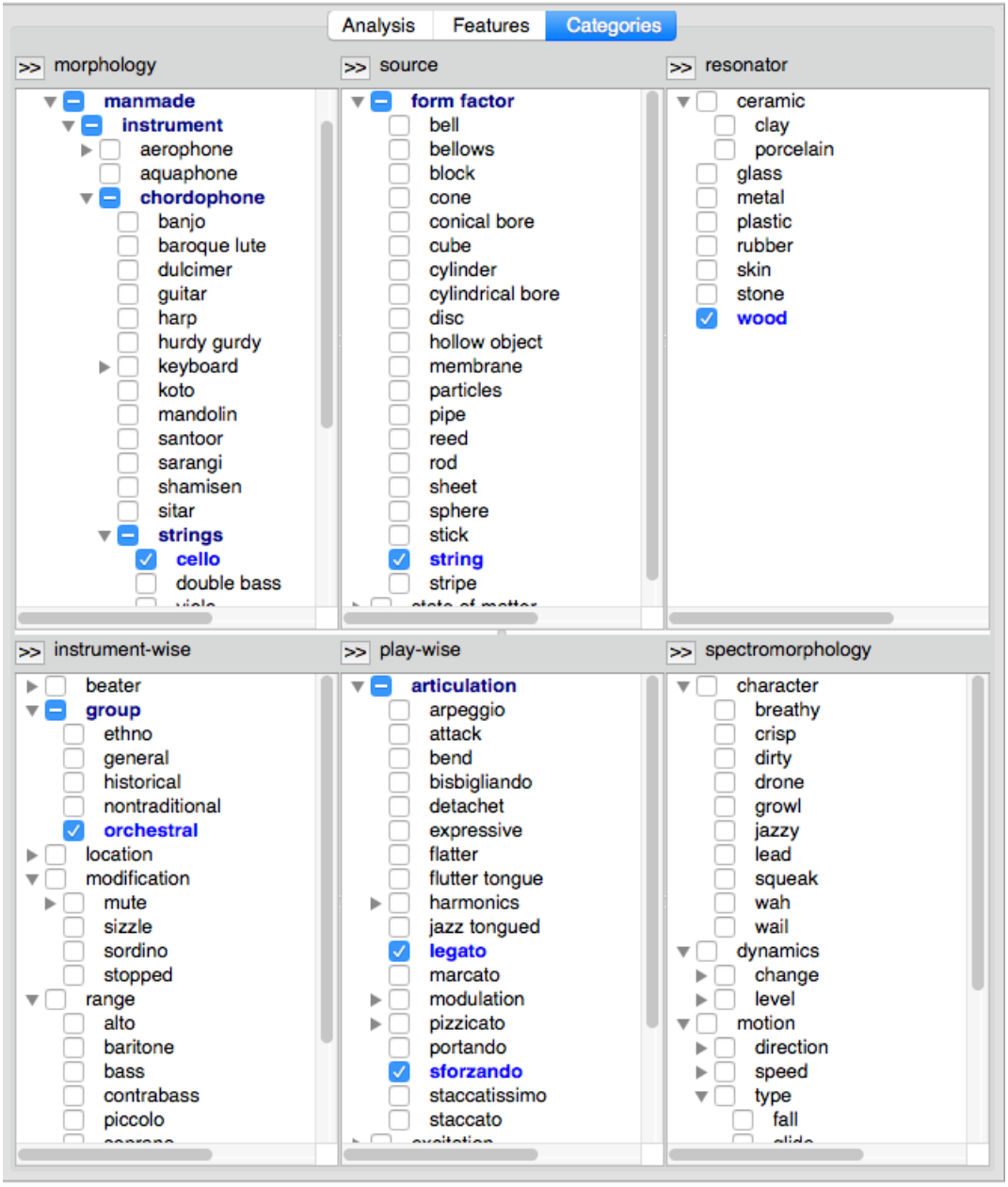


Figure 3.5: Category editor

The category editor also allows assigning of *imaginary categories* discussed in 2.4.3.3. Since an imaginary category is something associative rather than imperative it is also given a numeric measurement of strength x where $x \in (0, 1]$. The strength attribute allows to separate strong associations from weak ones and facilitates finer

degrees of control during search through imaginary categories. Figure 3.6 shows imaginary category (in green) assigned with associative strength of 0.5.

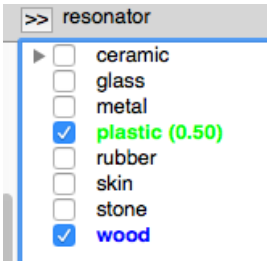


Figure 3.6: Imaginary category assignment

The category taxonomy can be completely redefined by user. The category editor allows renaming, adding or removing categories, including top-level ones.

3.1.4 Segmentation

Sound files that did not have accompanying segmentation files during database registration can be segmented with internal segmentation tools. In sEl a hybrid approach is adopted: first an element is processed by automatic segmentation[4], the results can then be interactively edited in the *Segment editor* shown in figure 3.7.

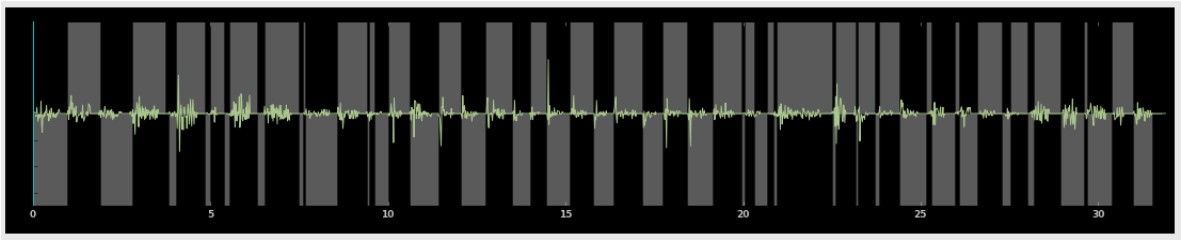


Figure 3.7: Segment editor window

Segments can be selected, merged, deleted and converted to database elements (Fig.3.8). Segmentation layout for each element is stored in a dedicated database table.

[4]Automatic segmentation is realized by multi-pass onset-based algorithm.

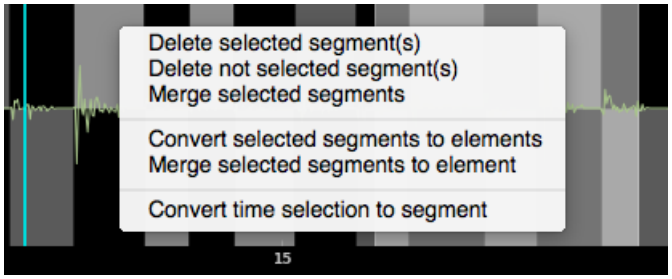


Figure 3.8: Segment editor commands

3.1.5 Auditioning

sEl offers an intuitive auditioning scheme while browsing sound files, elements and segments: audition can be initiated for a currently selected object and then automatically continued for other sound objects while the user keeps changing the selection. When several entities are selected simultaneously, audition will loop through them as if they were a single sound region. This method works throughout the system and allows quick evaluation of sound entities.

3.1.6 Creating database elements

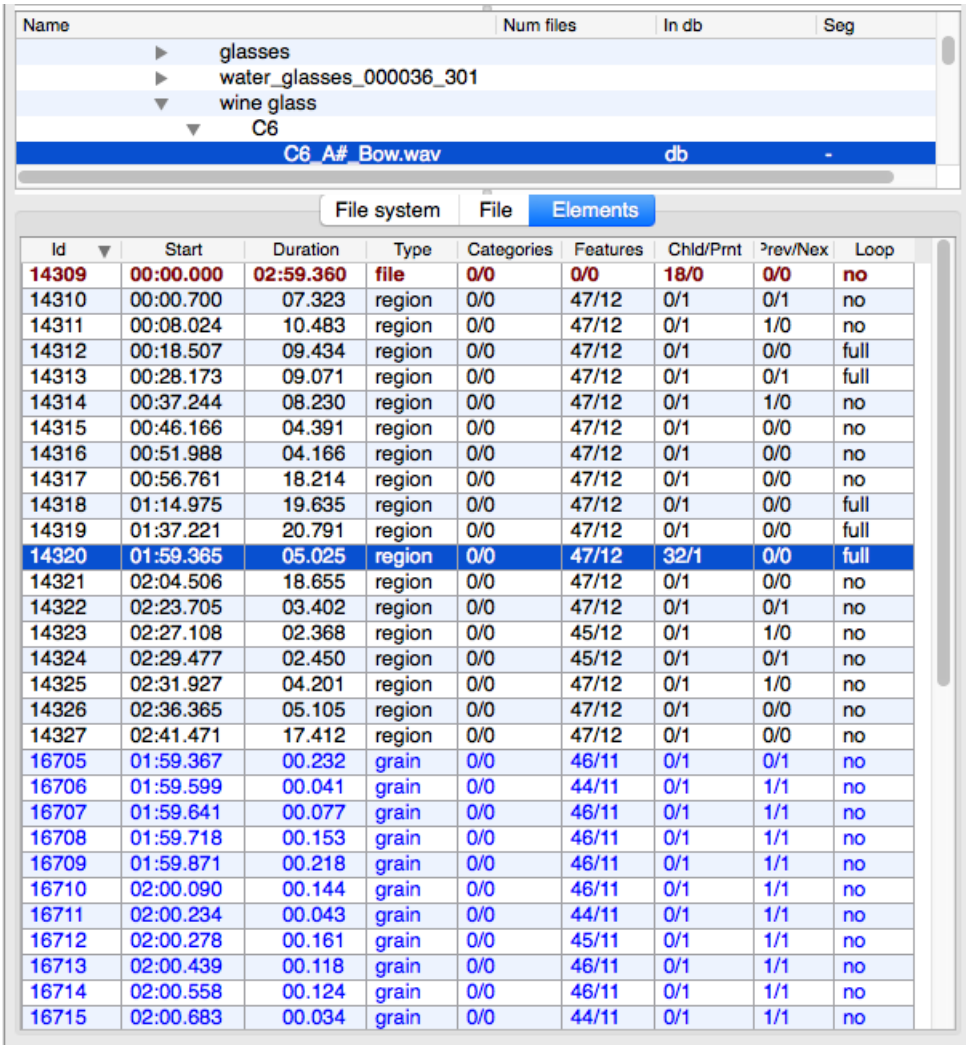
Segments contain information about temporal bounds of sound regions but nothing more. To be functionally useful, the segment has to be converted to a *database element* designed to hold all the semantics extracted from the corresponding sound region. There are two ways of how database elements can be created in sEl:

- automatic conversion of segments during sound file registration with database,
- interactive conversion of segments in *Segment editor* or *Element editor*.

3.1.7 Examining elements

Whenever a sound file is selected in the audio pool editor, if it is registered with the database its structural content will be retrieved and displayed in the *Element*

editor shown in figure 3.9. The element editor has a tabular layout in which each row corresponds to a single database element.



Id	Start	Duration	Type	Categories	Features	Chld/Prnt	Prev/Nex	Loop
14309	00:00.000	02:59.360	file	0/0	0/0	18/0	0/0	no
14310	00:00.700	07.323	region	0/0	47/12	0/1	0/1	no
14311	00:08.024	10.483	region	0/0	47/12	0/1	1/0	no
14312	00:18.507	09.434	region	0/0	47/12	0/1	0/0	full
14313	00:28.173	09.071	region	0/0	47/12	0/1	0/1	full
14314	00:37.244	08.230	region	0/0	47/12	0/1	1/0	no
14315	00:46.166	04.391	region	0/0	47/12	0/1	0/0	no
14316	00:51.988	04.166	region	0/0	47/12	0/1	0/0	no
14317	00:56.761	18.214	region	0/0	47/12	0/1	0/0	no
14318	01:14.975	19.635	region	0/0	47/12	0/1	0/0	full
14319	01:37.221	20.791	region	0/0	47/12	0/1	0/0	full
14320	01:59.365	05.025	region	0/0	47/12	32/1	0/0	full
14321	02:04.506	18.655	region	0/0	47/12	0/1	0/0	no
14322	02:23.705	03.402	region	0/0	47/12	0/1	0/1	no
14323	02:27.108	02.368	region	0/0	45/12	0/1	1/0	no
14324	02:29.477	02.450	region	0/0	45/12	0/1	0/1	no
14325	02:31.927	04.201	region	0/0	47/12	0/1	1/0	no
14326	02:36.365	05.105	region	0/0	47/12	0/1	0/0	no
14327	02:41.471	17.412	region	0/0	47/12	0/1	0/0	no
16705	01:59.367	00.232	grain	0/0	46/11	0/1	0/1	no
16706	01:59.599	00.041	grain	0/0	44/11	0/1	1/1	no
16707	01:59.641	00.077	grain	0/0	46/11	0/1	1/1	no
16708	01:59.718	00.153	grain	0/0	46/11	0/1	1/1	no
16709	01:59.871	00.218	grain	0/0	46/11	0/1	1/1	no
16710	02:00.090	00.144	grain	0/0	46/11	0/1	1/1	no
16711	02:00.234	00.043	grain	0/0	44/11	0/1	1/1	no
16712	02:00.278	00.161	grain	0/0	45/11	0/1	1/1	no
16713	02:00.439	00.118	grain	0/0	46/11	0/1	1/1	no
16714	02:00.558	00.124	grain	0/0	46/11	0/1	1/1	no
16715	02:00.683	00.034	grain	0/0	44/11	0/1	1/1	no

Figure 3.9: Database elements editor

Information being displayed contains:

- element's database id,
- temporal bounds (start time and duration),
- element's type,
- number of real and imaginary categories assigned to element,
- number of analyzed features (static and dynamic),

- element's containment relationships (number of child and parental elements),
- element's juxtaposition relationships (number of immediate preceding and following elements).

An example shown in the figure displays database elements for file `C6_A#_Bow.wav`. The element editor uses color coding to allow quick identification of parent elements (dark red) and child elements (blue) of a currently selected element.
It is in *Element editor* where analysis commands are issued with respect to selected elements.

3.1.8 Extracting features

To make elements compositionally useful they have to be analyzed for features. This can be done from either the Element editor or Audio pool editor by running analysis commands on the selection of files or elements. Analysis is done for those database features whose *active*[5] state is on. A reasonably rich roster of features (49) is offered by the system and it also can be expanded by user via

- constructing new analysis graphs from the system library's processing units and/or
- writing new feature extractor classes via inheriting from ProcessingNode class[6].

Each feature extracted from the database element becomes a dimension along which elements can be compared and the distance between them can be computed. Having nearly fifty extractable features allows the establishment of a high-dimensional EncycloSpace available for compositional navigation. As mentioned above, it is possible to vary the dimensionality of EncycloSpace by adding new features or disabling existing ones.

[5]Feature activity status is stored in its descriptor database record (see 3.8) and can be controlled by user.

[6]ProcessingNote is a base class from which all analysis units of the system are created.

3.1.9 Recombining elements

The availability of multi-dimensional EncycloSpace makes it a perfect site for applying compositional strategies. The space can be navigated by search queries driven by specific sets of constraints. Execution of such a process yields a collection of elements responding to those constraints. Thus obtained, collections become recombined compositional subspaces — intentional extractions from the EncycloSpace. Such subspaces can be further recombined by creating collections of collections.

The collection has double status in sEl: on one hand it is an aggregate entity consisting of multiple objects, on the other hand it is a single object which can be manipulated as such by compositional routines.

Composing collections is an important part of the workflow which can be referred to as *instrumentation phase*.

3.1.10 Composing with collections

When necessary collections are assembled the further step is putting them into temporal and vertical relationships. This can be done by using compositional objects of sEl which offer a wealth of methods to unwrap potential richness of aggregate sound entities. The compositional use of collections is described in more detail in 3.4.5.4.

3.2 Database schema

sEl's database schema corresponds to the three-partite ontological layout of the system's EncycloSpace and contains:

- Material layer (neutral)
- Semantics layer (esthesis)
- Recombinant layer (poiesis)

3.2.1 Material layer

The *Material layer* of the database schema consists of relations that store a topology of sound objects in the system. Its *entity/relationship* configuration is shown in figure 3.10[7]. The material layer comprises the following relations:

- dbDataFile (table *datafile*) — information about files in Audio pool,
- dbDataFileGroup (table *datafile_group*) — information about file groups,
- dbElement (table *element*) — information about specified sound file regions,
- dbElementParent (table *elem_parent*) — information about elements containment in larger elements,
- dbElementNext (table *elem_next*) — information about elements temporal juxtapositioning,
- dbElementSegmentation (table *element_segmentation*) — information smaller divisions within element.

3.2.1.1 dbDataFile

This ORM class (and underlying SQL table *datafile*) contains information about data files stored in the scope of the system visibility (currently *Audio pool*[8]). Its structure is shown in table 3.1.

3.2.1.2 dbDataFileGroup

dbDataFileGroup (SQL table *datafile_group*) is a lightweight construct that describes file groups (table 3.2). File group is essentially a folder where the file is contained. In sEl folders can have additional semantics. For instance folder containing samples of a single instrument has semantics of *sampleset*. This information helps sEl to fine-tune processing of the folder's content.

[7]Entities in the diagram are labeled with ORM classes names as defined in the program code
[8]In the future the visibility can be expanded to include more media types

Table 3.1: **dbDataFile** (table *datafile*)

Column name	Type	Key	Comments
file_id	integer	PK	unique file identifier
group_id	integer	FK	file group identifier (from table 3.2)
name	varchar(128)		file name
path	varchar(256)		relative path to the file with respect to Audio pool
type	char(10)		sound, graphics, video etc. Non audio types are present to accommodate future expansion of the system into media types other than sound.
subtype	char(16)		waveform, feature, segmentation etc
format	char(4)		media file format, i.e. wav, aiff, mp4 etc
annotation	varchar(128)		comments
datetime	timestamp		file creation date and time
duration	real		data duration in seconds
num_channels	smallint		number of data channels
num_frames	integer		number of data frames in the file
frame_rate	real		sampling rate, frame rate for multi-frame file types
bit_depth	smallint		number of bits per data value of a single channel
data_type	char(10)		type of data values stored in the file

Table 3.2: **dbDataFileGroup** (table *datafile_group*)

Column name	Type	Key	Comments
grp_id	integer	PK	unique file group identifier
type	varchar(32)		collection, sampleset etc
name	varchar(250)		group's path with respect to Audio pool

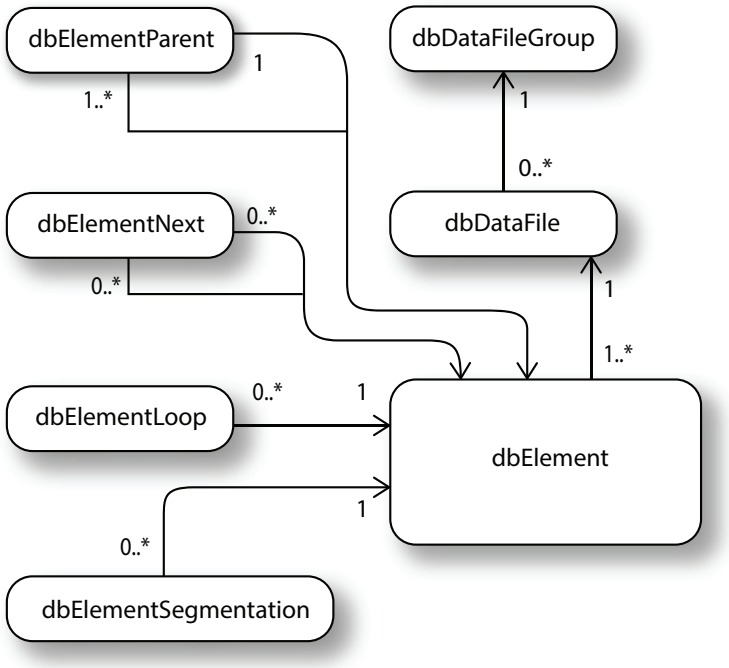


Figure 3.10: Material layer of the database schema

3.2.1.3 dbElement

Class dbElement (SQL table *element*) is a central component not only of the *Material layer* but also a main communication channel among all three layers of the database (as we will soon see). The database element represents the navigational entry inside EncycloSpace. All together, these entries constitute the navigable space of the system.

Table 3.3: **dbElement** (table *element*)

Column name	Type	Key	Comments
el_id	bigint	PK	unique element identifier
root_id	bigint	FK	id of root element (from 3.3. Root element is an element that represents whole file)
type	integer		element type
file_id	integer	FK	element's file id (from 3.1)
start_time	real		start time within the file (in seconds)
end_time	real		end time within the file (in seconds)

3.2.1.4 dbElementParent

Elements can have hierarchical relationships of containment. However SQL syntax lacks appropriate specifications. Class dbElementParent (SQL table *elem_parent*) is a way to circumvent this limitation by storing containment information in a separate table.

Table 3.4: **dbElementParent** (table *elem_parent*)

Column name	Type	Key	Comments
chld_id	bigint	FK	child element id (from 3.3)
parent_id	bigint	FK	parent element id (from 3.3.

3.2.1.5 dbElementNext

An element can be related to other elements with which it shares a common temporal border, i.e. immediately preceding or following them in the sound file. This information can be valuable whenever element concatenation is considered, because it allows the singling out of concatenation cases not requiring processing due to the elements' natural order.

Table 3.5: **dbElementNext** (table *elem_next*)

Column name	Type	Key	Comments
prev_id	bigint	FK	preceding element id (from 3.3)
next_id	bigint	FK	subsequent element id (from 3.3.

3.2.1.6 dbElementLoop

An element can have internal time points within which it can be seamlessly looped. This information can help to determine whether the element can be performed in a sustained manner.

Table 3.6: **dbElementLoop** (table *loop*)

Column name	Type	Key	Comments
loop_id	integer	PK	unique loop id)
elem_id	bigint	FK	id of containing element (from 3.3)
start_time	real		loop's starting time within element
end_time	real		loop's end time within element
reversible	boolean		whether loop points should wrap or bounce
full_loop	boolean		whether whole element can be looped

3.2.1.7 dbElementSegmentation

An element, if it is not too short, can be a subject of segmentation. dbElementSegmentation table stores information about elements' segments.

Table 3.7: **dbElementSegmentation** (table *element_segmentation*)

Column name	Type	Key	Comments
el_id	bigint	FK	id of element (from 3.3) which owns the segmentation
datetime	timestamp		segmentation creation date and time
segments	real[]		segments data

3.2.2 Semantics layer

The *Semantics layer* contains a knowledge base extracted from entities stored in the *Material layer*. This knowledge base is organizationally structured as an ER layout shown in figure 3.11. If the material layer represents navigational entries into the EncycloSpace, the semantics layer betokens its informational body. Or in semiotics' terms: with respect to EncycloSpace, the semantics layer is collection of *signifier* entities while the material layer is an assemblage of *signified* entities.

Semantics layer consists of the following ORM classes and SQL relations:

- dbDescriptor (table *descriptor*) — contains semantics layer's taxonomy: descriptions of categories, features and tags,
- dbClass (table *class*) — information about descriptor hierarchies,
- dbSymbol (table *symbol*) — mapping between symbol names and integer ids by which they are represented elsewhere in the system,
- dbFeatureStats (table *feature_stats*) — feature related statistics used for data normalization during analysis,
- dbElementCategory (table *elem_cat*) — information about assignment of categories to elements,
- dbImaginaryCategory (table *imagine*) — information about assignment of imaginary categories to elements,
- dbAnalysisConfig (table *analysis_config*) — information about analysis graphs configurations used to analyze features,
- dbAnalysisConfigFeature (table *analysis_config_feature*) — information about analysis graphs' feature capabilities,
- dbElementFeature (table *element_feature*) — mean value of extracted feature.
- dbElementFeatureCV (table *element_feature_cv*) — stores temporal behavior of dynamic features, i.e. those that change over time.

3.2.2.1 dbDescriptor

dbDescriptor (SQL table *descriptor*) stores the system's vocabulary of categories and features, i.e. its descriptive lexicon. The structure is shown in table 3.8.

3.2.2.2 dbClass

The Categories taxonomy forms a hierarchical tree-like structure. Except for the top level, each category down the tree becomes part of the lineage of inheritance. Direction down the tree exemplifies specialization, direction up the tree - generalization.

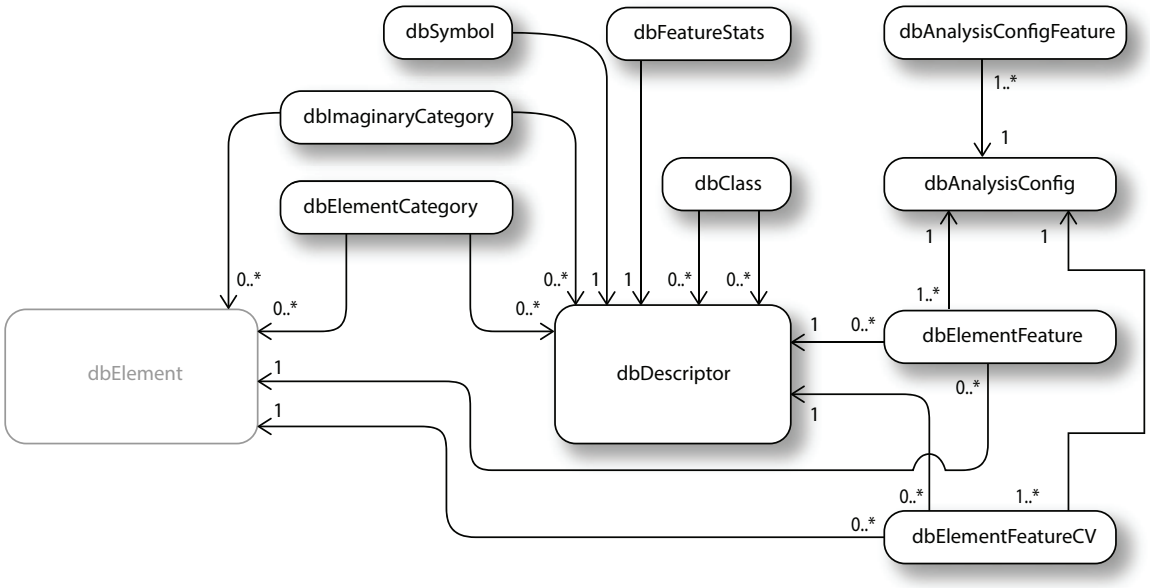


Figure 3.11: Semantics layer of the database schema

Table 3.8: **dbDescriptor** (table *descriptor*)

Column name	Type	Key	Comments
desc_id	integer	PK	unique descriptor id
parent_id	integer	FK	id of immediate parent (from 3.8)
desc_name	varchar(48)		descriptor name
desc_type	varchar(10)		descriptor type: category, feature or tag
data_type	varchar(32)		descriptor data type: boolean, integer, real, symbol, text, array or symbol_array
tree_level	smallint		a level down the hierarchy of containment
dynamic	boolean		whether descriptor represents feature that changes over time and thus requires analysis of temporal behavior
active	boolean		pertains only to features and indicates whether feature should be analyzed or skipped

Class membership is promoted in the direction of generalization with inclusion of all the tree nodes encountered in that direction. An example in figure 3.12 shows a morphology generalization path for french horn with class membership in each layer above.

Adding new category to the taxonomy spawns a generalization chain of class memberships all the way to the top. All class memberships are stored in dbClass relation (table 3.9).

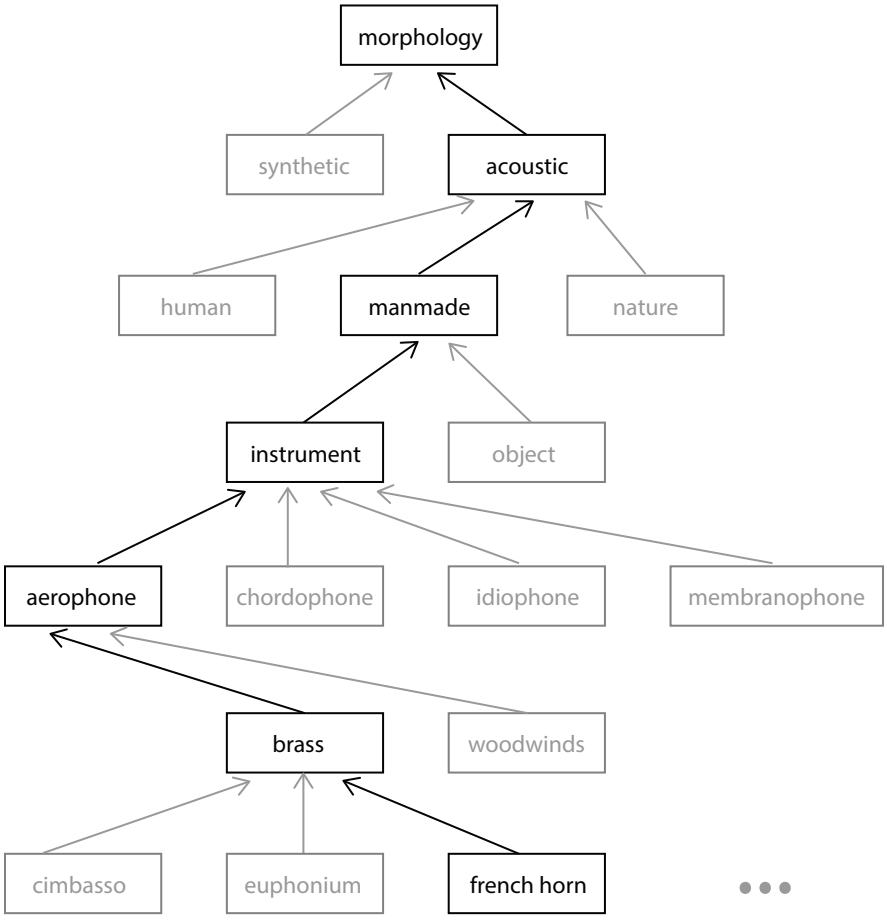


Figure 3.12: Categoric generalization of french horn

3.2.2.3 dbSymbol

Symbolic descriptors[9] need to dereference their numeric values into human readable form. Relation dbSymbol is used to store those lookup tables. The structure is shown in table 3.10.

[9]Symbolic descriptors are those whose *data_type* field is *symbol* or *symbol_array*

Table 3.9: **dbClass** (table *class*)

Column name	Type	Key	Comments
desc_id	integer	FK	descriptor id (from 3.8)
class_id	integer	FK	id of immediate parent (from 3.8)
type	varchar(8)		differentiates between user created memberships and automatically generated ones for closure
distance	smallint		tree distance between category and class

Table 3.10: **dbSymbol** (table *symbol*)

Column name	Type	Key	Comments
sym_id	integer	PK	unique symbol id
desc_id	integer	FK	id of descriptor (from 3.8) which uses this symbol as its value
sym_name	varchar(48)		symbol name

3.2.2.4 dbFeatureStats

In the course of feature analysis the results need to be normalized for compatibility during element distance computation. Relation dbFeatureStats stores necessary feature-wise statistics which are used during normalization.

Table 3.11: **dbFeatureStats** (table *feature_stats*)

Column name	Type	Key	Comments
desc_id	integer	FK	feature id (from 3.8)
min	real		minimum value of feature across the database
max	real		maximum value of feature across the database
mean	real		average value of feature across the database
std	real		standard deviation of feature across the database

3.2.2.5 dbElementCategory

This relation stores information about elements category membership. Each assigned category automatically promotes that membership to include the category's generalization chain (see 3.2.2.2).

Table 3.12: **dbElementCategory** (table *elem_cat*)

Column name	Type	Key	Comments
el_id	bigint	FK	element id (from 3.3)
desc_id	integer	FK	feature id (from 3.8)

3.2.2.6 dbImaginaryCategory

This relation stores information about elements category membership. Each assigned category automatically promotes that membership to include the category's generalization chain (see 3.2.2.2).

Table 3.13: **dbImaginaryCategory** (table *imagine*)

Column name	Type	Key	Comments
el_id	bigint	FK	element id (from 3.3)
desc_id	integer	FK	feature id (from 3.8)
strength	real		associative strength of the category assignment $\in (0, 1]$

3.2.2.7 dbAnalysisConfig

To undertake feature analysis of an element first analysis graph has to be constructed. Analysis graph is a configuration of processing pipeline(s) which interconnect analysis algorithms in order to extract specific features. Analysis graph's component algorithms receive specific parameter settings that influence various aspects of feature computation. Since different settings of the same algorithms topology can produce

different results we need to store an exact fingerprint of the analysis graph that could be used as an identifier for specific feature analysis. Moreover we need to store analysis graph in the form which can be later deserialized to reconstruct the graph in its entirety. Relation dbAnalysisConfig stores that information.

Table 3.14: **dbAnalysisConfig** (table *analysis_config*)

Column name	Type	Key	Comments
graph_id	integer	PK	unique id of analysis graph
config	varchar(3000)		analysis graph encoded as text string from which it can be deserialized
config_id	integer		checksum of config string
date_added	timestamp		creation date and time of analysis graph

3.2.2.8 dbAnalysisConfigFeature

Mapping between analysis graph and feature is *one-to-many*, i.e. a single graph can compute more than one feature. Relation dbAnalysisConfigFeature stores information about analysis graphs' feature computation capabilities.

Table 3.15: **dbAnalysisConfigFeature** (table *analysis_config_feature*)

Column name	Type	Key	Comments
graph_id	integer	FK	id of analysis graph (from 3.14)
desc_id	integer	FK	feature id (from 3.8)

3.2.2.9 dbElementFeature

dbElementFeature stores a single feature's value extracted by analysis. This value is averaged over the course of the element's duration and thus represents temporally aggregated perspective of the element. Since SQL has strongly typed field syntax and since different features can be represented by different data types we cannot just use

one field to store any type of data. Instead we reserve fields of several types to store any expected feature values.

Table 3.16: **dbElementFeature** (table *element_feature*)

Column name	Type	Key	Comments
el_id	bigint	FK	element id (from 3.3)
desc_id	integer	FK	feature id (from 3.8)
graph_id	integer	FK	id of analysis graph (from 3.14)
datetime	timestamp		feature computation date and time
int_val	integer		integer value
real_val	real		floating point value
text_val	varchar(64)		textual value
array_val	real[]		array value

3.2.2.10 dbElementFeatureCV

For features that change over time, e.g. pitch, loudness, spectral characteristics and alike, we need to compute a set of scalar data that could sufficiently capture temporal behavior. A diversified set of parameters for exactly this purpose is described in chapter 11 of [100] where it is called *characteristic values*. With some reductions we adopt this set. Characteristic values of dynamic features are stored in dbElementFeatureCV table. The aforecited source can be consulted for the details on computation formulae. The structure is shown in table 3.17.

3.2.3 Recombinant layer

The *Recombinant layer* is directly related to the compositional function of the system for it stores information about recombinations of the elements produced in the course of various compositional strategies execution. In the *ontological communication schema* with respect to EncycloSpace this layer epitomizes a material deployment site for the *poiesis* channel — an accretion of the navigable space via productive effort. Entities stored in *recombinant layer* are normally produced via use of both *material*

Table 3.17: **dbElementFeatureCV** (table *element_feature_cv*)

Column name	Type	Key	Comments
el_id	bigint	FK	element id (from 3.3)
desc_id	integer	FK	feature id (from 3.8)
graph_id	integer	FK	id of analysis graph (from 3.14)
datetime	timestamp		feature computation date and time
geomean	real		geometric mean
std	real		standard deviation
startval	real		starting value
endval	real		closing value
minval	real		minimum value
maxval	real		minimum value
absrange	real		absolute range
slope	real		slope of linear approximation
curve	real		curve of 2nd order polynomial approximation
t_mean	real		temporal centroid
t_antimean	real		temporal anticentroid
t_std	real		temporal standard deviation
t_skew	real		temporal skewness
t_kurt	real		temporal kurtosis
t_atk	real		AR attack time
t_rel	real		AR release time
t_invatk	real		inverse AR attack time
t_invrel	real		inverse AR release time
s_mean	real		spectral average
s_std	real		spectral standard deviation
s_skew	real		spectral skewness
s_kurt	real		spectral kurtosis
s_band0	real		spectral energy in $[0, 1]$ Hz band
s_band1	real		spectral energy in $[1, 10]$ Hz band
s_band2	real		spectral energy in $[10, 20]$ Hz band
s_band3	real		spectral energy in $[20, 40]$ Hz band
s_band4	real		spectral energy in $[40, 100]$ Hz band

and *semantics* layers. However as far as database schema, the recombinant layer only has relations to the material layer as shown in figure 3.13.

Recombinant layer consists of the following ORM classes and SQL relations:

- dbCollection (table *collection*) — stores metadata of element collections,
- dbCollectionElements (table *collection_elements*) — stores collections content,
- dbSequenceElements (table *sequence_elements*) — stores ordered collections content.
- dbCollectionGroup (table *collection_group*) — stores collection groups content,

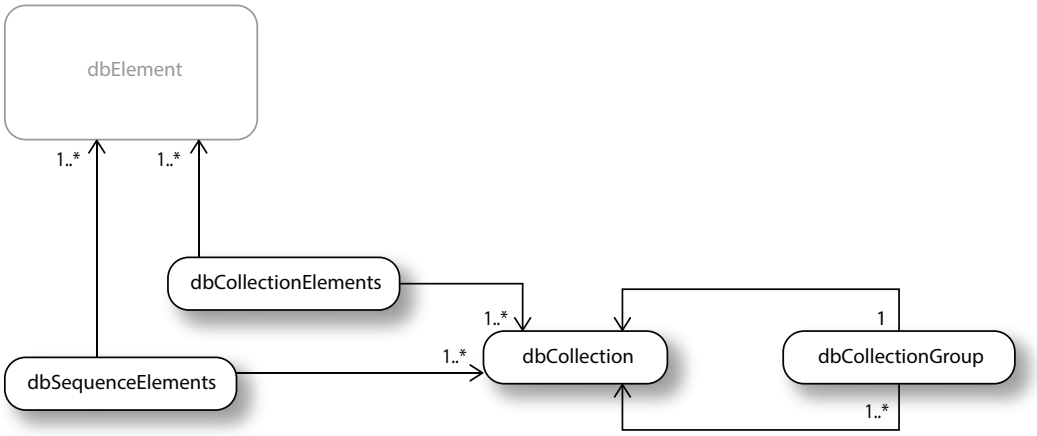


Figure 3.13: Recombinant layer of the database schema

3.2.3.1 dbCollection

Collections are core elements of recombinant composition. Collection is a result of an EncycloSpace navigation which extracts its parts under particular strategy and saves them as a new entity. ORM class dbCollection and its underlying SQL table store meta information about collections.

Table 3.18: **dbCollection** (table *collection*)

Column name	Type	Key	Comments
coll_id	integer	PK	unique collection id
coll_name	varchar(48)		collection name
coll_type	varchar(10)		collection type: set, sequence or group
query_context	varchar(4096)		database query context which produced the collection
description	varchar(128)		

3.2.3.2 dbCollectionElements

This relation defines content of unordered collections by mapping elements to collections.

Table 3.19: **dbCollectionElements** (table *collection_elements*)

Column name	Type	Key	Comments
coll_id	integer	FK	collection id (from 3.18)
el_id	bigint	FK	element id (from 3.3)

3.2.3.3 dbSequenceElements

Collections can be unordered (*set* type) or ordered (*sequence* type). dbSequenceElements is the same as dbCollectionElements except that it also stores the elements' order.

Table 3.20: **dbSequenceElements** (table *sequence_elements*)

Column name	Type	Key	Comments
coll_id	integer	FK	collection id (from 3.18)
el_id	bigint	FK	element id (from 3.3)
el_ndx	integer		element index within sequence

3.2.3.4 dbCollectionGroup

One of the strongest features of collections is that they can contain other collections. This kind is called a *collection group* which is a collection with *type* field set to *group*. Self-referencing capability makes collections a powerful recombinant tool. Relation dbCollectionGroup is used to map collections to groups.

Table 3.21: **dbCollectionGroup** (table *collection_group*)

Column name	Type	Key	Comments
group_id	integer	FK	group collection id (from 3.18)
coll_id	integer	FK	collection id (from 3.18)

sEl's database schema with all layers assembled is shown in figures 3.14 and 3.15.

3.3 Analysis subsystem

Involvement of sampled sound semantics in the compositional process dictates the necessity of a dedicated environment for administering audio information retrieval. The analysis subsystem of sEl provides such an environment implemented with the following goals in mind:

- Compositional perspective on selecting feature set
- Integration of tools
- Flexibility
- Visualization
- Extensibility

3.3.1 Compositional perspective on feature set

In sEl feature analysis has no bigger purpose than guiding compositional process. The software comes with a default roster of 49 analyzable features (presented below)

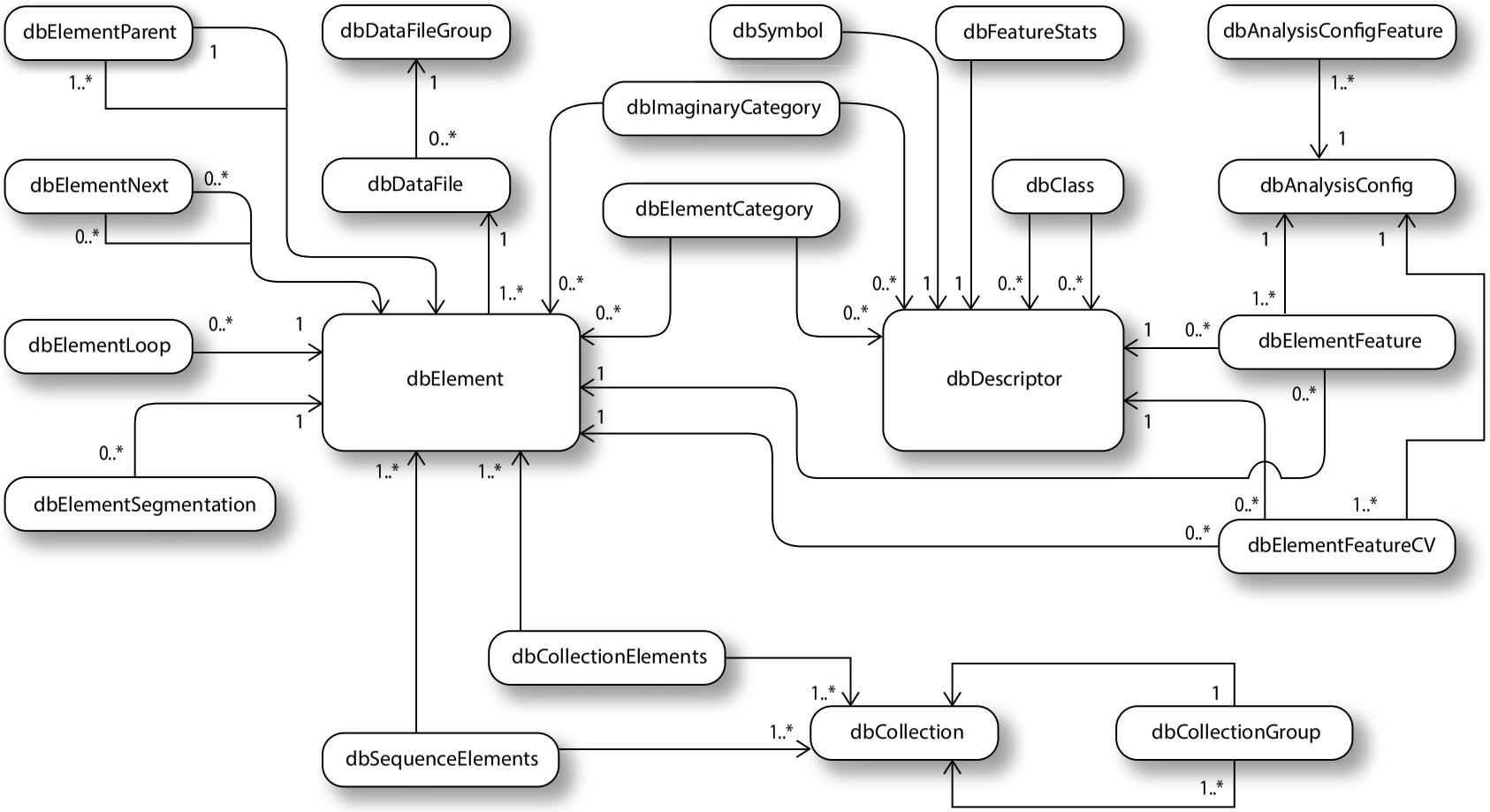


Figure 3.14: ER schema of sEl database.

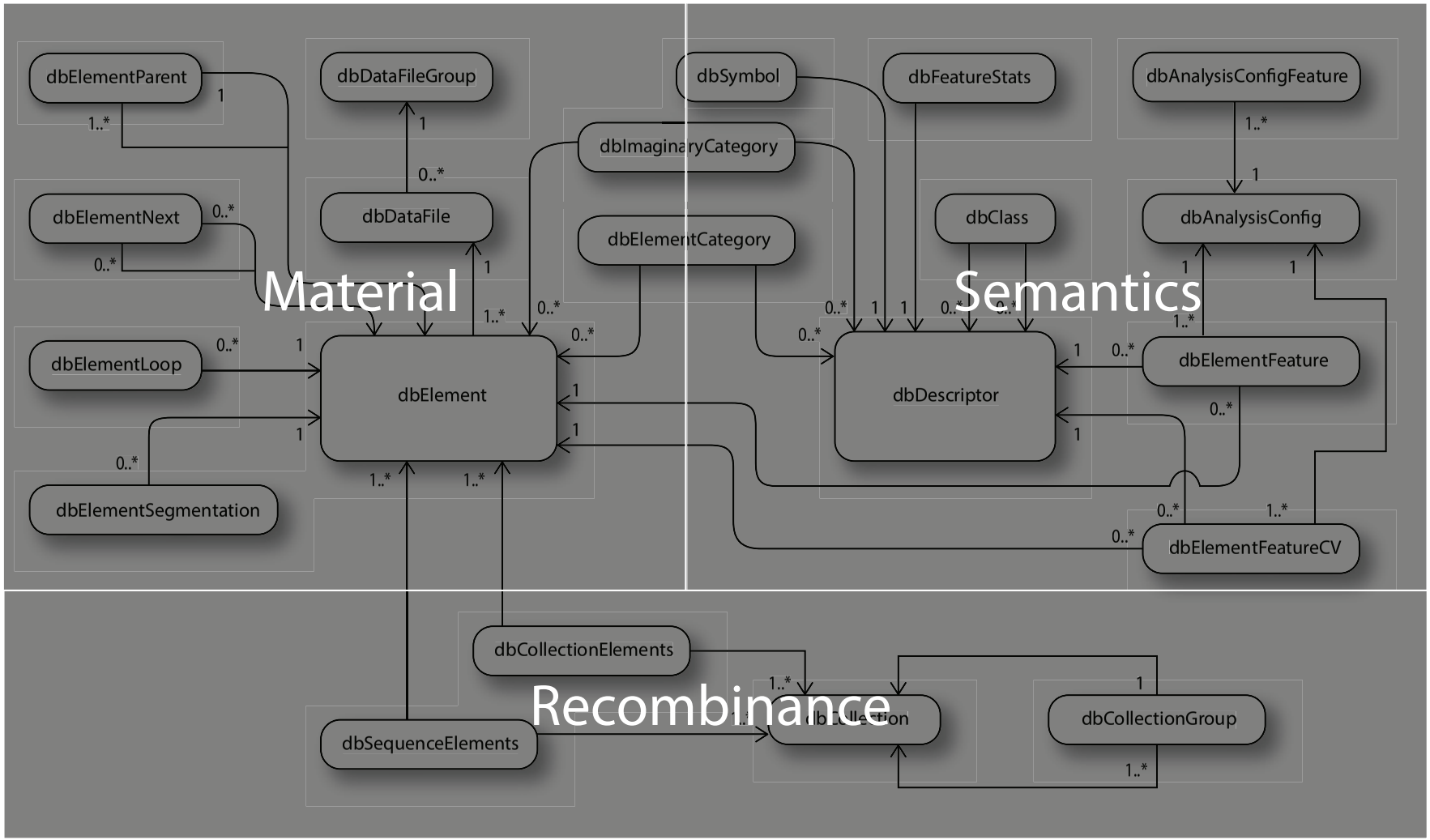


Figure 3.15: ER schema of sEl database with semantic areas.

that span most of the salient feature groups. The choice of default features was in large part informed by studying related work and literature in music informatics on audio feature sets.

Audio feature design is a separate discipline which is still far from reaching a conclusive state. As pointed out in [62], contributions to this discipline are dominated by "hand-crafted design" ingrained into "shallow architectures" that overlook "compositional containment hierarchy" pertaining to music. Even though the motivation for a "deep architecture" approach to audio feature design has been expressed no actual implementation has been produced as of yet.

An additional challenge for omni-source systems like ours is to avoid features that imply restrictions on sound. For instance *ADSR envelope* anticipates that sound amplitude fits into an *attack-decay-sustain-release* template which many sounds will not comply with — therefore this feature cannot not be included in sEl feature set.

Intended as a full time-scale system (see 2.1.1.1) sEl requires feature design to account for multi-scale signal representations. This sort of problematics is recognized in [57]:

> Using representations at multiple scales allows much flexibility to model the structure of the data. Multi-scale representations offer a natural way to jointly model local and global aspects, without having prior knowledge about the local and global structures.

In sEl problem of multi-scale representation is addressed by applying pre-processing and multi-pass analyses aiming to adapt the analysis pipeline to process sounds with arbitrary temporal and spectral characteristics.

Another facet of compositional perspective on features is making them represented by psychometric units rather than physical ones (2.4.2.3). This allows to bring compositional planning as close as possible to the listening experience. However, in order to aid switching between psychometric and physical units when necessary, sEl provides a set of converting routines.

Modifying audio feature taxonomy facets, laid out in [42], [89] and [100], towards compositional use sEl combines features into following groups:

- Temporal descriptors
- Energy descriptors
- Tonal descriptors
- Rhythm descriptors
- Spectral descriptors

Following sections describe each descriptor group. All descriptors are also presented in a table form in Appendix A.

3.3.2 Temporal descriptors

Group of temporal features includes characteristics of horizontal structure and variability of sound. Following descriptors are included in the temporal group:

- Timescale
- Event cardinality
- Temporal density
- Onset cardinality
- Dynamic complexity
- Inter-onset intervals
- Q-time

3.3.2.1 Timescale

This descriptor characterizes timescale class of sound elements following the taxonomy shown in table 3.22. The taxonomy utilizes Western music durational terminology to make this descriptor more intuitive. Each duration class is defined as range with minimum duration corresponding to tempo of 160 pbm (ca *allegro*) and maximum

corresponding to 80 bpm (ca *adagio*). Not by any stretch these divisions are intended as definitions but simply as intuitively understandable demarkations.

Table 3.22: **Timescale descriptor ranges**

Id	Symbol	Range
0	64th	$\in [23, 46)ms$
1	32nd	$\in [46, 94)ms$
2	16th	$\in [94, 188)ms$
3	quaver	$\in [188, 375)ms$
4	crotchet	$\in [375, 750)ms$
5	2nd	$\in [0.75, 1.5)sec$
6	whole	$\in [1.5, 3)sec$
7	double	$\in [3, 6)sec$
8	long	$\in [6, 12)sec$
9	system	$\in [12, 48)sec$
10	page	$\in [48sec, 3.2min)$
11	section	$\in [3.2, 12.8)min$
12	piece	$\in [12.8, 51.2)min$
13	concert	$\in [51.2min, \infty)$

3.3.2.2 Event cardinality

Omni-source environment of sEl anticipates sound elements with arbitrary content. One of the most general structural characteristics is containment degree, or the number of internal events, i.e. non-silent chunks divided by regions of silence. *Event cardinality* represents such a quantity.

3.3.2.3 Temporal density

Temporal density is calculated as a ratio of sound's non-silent portion to its overall duration:

$$D_T = \frac{L_{non-silent}}{L_{total}} \tag{3.1}$$

3.3.2.4 Onset cardinality

Onsets mark perceptual non-gradual changes discernible in the stream of sound. Number of such changes is saved as *onset cardinality* descriptor which characterizes structural complexity of sound elements on a finer level than that of event cardinality. Onset detection can be approached by broad category of methods focused on monitoring different characteristics of sound and combining those in a number of ways. Many onset methods are developed for specific types of sound like singing voice [113] or distinctly rhythmic formations [60]. Such methods would not be very useful for omni-source application like sEl. It requires general solution for onset detection like the ones described in [46] or [25]. In sEl we use methods designed in [25] which are implemented as a C library with the Python front end[24].

3.3.2.5 Dynamic complexity

Music complexity is explored at depth in [111] where it is defined as

> a high-level, intuitive attribute, which can be experienced directly or indirectly by the active listener, so it could be estimated by empirical methods. In particular we define the complexity as that property of a musical unit that determines how much effort the listener has to put into following and understanding it.

Dynamic complexity is a descriptor that evaluates *dynamic component* of *acoustic complexities* via algorithm presented on page 24 of [111] with concluding formula:

$$C_{dynI} = \frac{1}{M} \sum_{i=0}^{M-1} |V_{dB}(i) - L| \tag{3.2}$$

where M is number of loudness measurements over the course of the sound element, $V_{dB}(i)$ — individual loudness measurements and L is a weighted average thereof.

The signal is pre-filtered to approximate B-weighting curve to account for perceptual non-uniformity of human ear.

3.3.2.6 Inter-onset intervals

Sequence of time intervals between neighboring onsets characterizes the evolution of rate of change as the element progresses through time. This characteristic can vary within a considerable range of values. Thus we store it not as an averaged scalar value but as a dynamic feature described by a set of statistical characteristics (3.2.2.10).

3.3.2.7 Q-time

This descriptor is essentially an attack time, i.e. the location of a time point perceived as the beginning of sound which normally coincides with the maximum of energy after the start of an element. This descriptor is called *Q-time* to emphasize its role in temporal placement of elements during *temporalization transformation* (see 2.5.2.2). Temporalization defines time locations at which elements should be deployed as a way to structurize psychological (musical) time. Difference between naive and q-time guided temporalizations can be observed by comparing figures 3.16 and 3.17. Both figures contain placement of three elements at intended time locations marked *T1, T2* and *T3* (in black color). Time points labeled with red *Q* show q-time locations within each element and red *T1, T2* and *T3* mark the time points where elements start times will be perceived.

In figure 3.16 elements placement is done with respect to their physical start which leads to perceptual distortion of temporalization (note the mismatch between intended and perceptual locations). In figure 3.17 the elements are placed with respect to their q-time descriptors which aligns intended and perceptual time points.

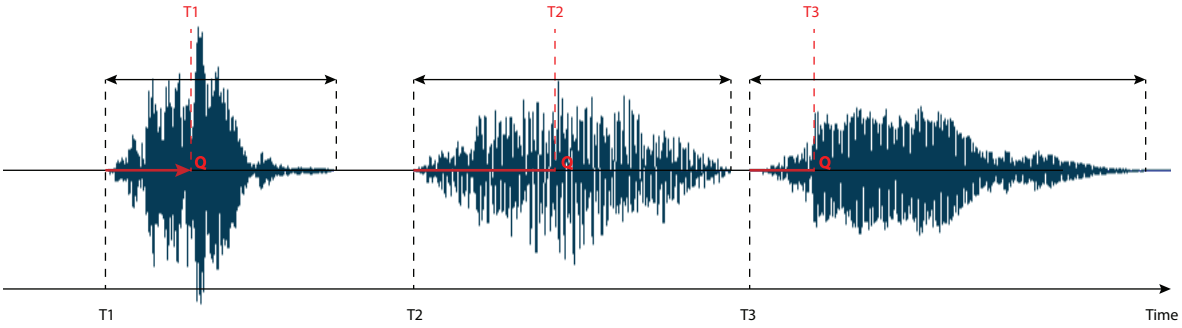


Figure 3.16: Naive elements temporalization

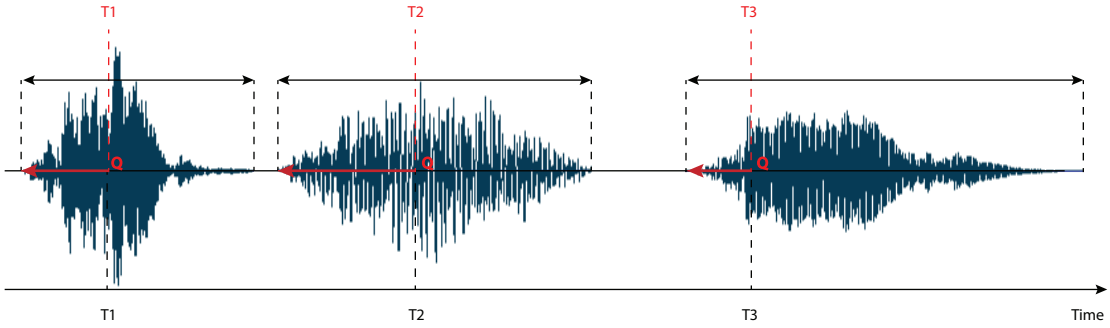


Figure 3.17: Q-time guided elements temporalization

3.3.3 Energy descriptors

This group of descriptors combines characteristics derived from aspects related to sound energy. It includes following features:

- Dynamics
- ELF Loudness
- Bark bands energy
- Spectral RMS

3.3.3.1 Dynamics

This descriptor represents discrete groups of loudness values mapped to music dynamics symbols. The groups use perceptually homogeneous partitioning with each group occupying 10 decibels as shown in table 3.23. As in *timescale* descriptor (3.3.2.1) this is not a definition of music symbols but a convenient subdivision of the loudness con-

tinuum allowing quick selection of volume ranges. Step size of 10 decibels is chosen as an approximate loudness doubling/halving interval [73] so that neighboring groups are twice as loud or soft. *Dynamics* is a *global descriptor*, i.e. its value describes signal as a whole.

Table 3.23: **Dynamics descriptor ranges**

Id	Symbol	Range (dBFS)
0	ppppp	$\in [-\infty, -90)$
1	pppp	$\in [-90, -80)$
2	ppp	$\in [-80, -70)$
3	pp	$\in [-70, -60)$
4	p	$\in [-60, -50)$
5	mp	$\in [-50, -40)$
6	mf	$\in [-40, -30)$
7	f	$\in [-30, -20)$
8	ff	$\in [-20, -10)$
9	fff	$\in [-10, 0)$

3.3.3.2 ELF Loudness

ELF loudness describes evolution of the perceived loudness over the course of the element. *ELF* stands for *equal-loudness filtered* because before the loudness is measured the signal is filtered by an approximated inversion of Fletcher-Munson equal-loudness contour. This transfers the signal into psychometric domain where frequency-related non-uniformity of loudness perception is eliminated. As *dynamics* this feature is also represented in dBFS units.

3.3.3.3 Bark bands energy

Compared to the previous two descriptors *Bark bands energy* is a step towards vertical, i.e. spectral, specificity. Whereas *dynamics* provides an aggregated view on loudness and *ELF loudness* traces loudness evolution, *Bark bands energy* measures

sound energy in frequency terms related to human hearing. The *bark scale* proposed in 1961 by Eberhard Zwicker [126] divides the audible range into juxtaposed frequency intervals corresponding to the cochlear critical bands of frequency discrimination. Zwicker's divison defines 24 bands covering frequency range of $[20, 15500]Hz$. sEl uses slightly expanded set of 27 bands covering range of $[0, 22050]Hz$. Table 3.24 shows bands ranges (with three non-Zwicker bands printed in italics).

Table 3.24: **Bark bands energy descriptor frequency ranges**

Band id	Center frequency, Hz	Cut-off frequency, Hz	Bandwidth, Hz
		0	
1	*25*	*50*	*50*
2	75	100	50
3	*125*	*150*	*50*
4	175	200	50
5	250	300	100
6	350	400	100
7	455	510	110
8	570	630	120
9	700	770	140
10	845	920	150
11	1000	1080	160
12	1175	1270	190
13	1375	1480	210
14	1600	1720	240
15	1860	2000	280
16	2160	2320	320
17	2510	2700	380
18	2925	3150	450
19	3425	3700	550
20	4050	4400	700
21	4850	5300	900
22	5850	6400	1100
23	7050	7700	1300
24	8600	9500	1800
25	10750	12000	2500
26	13750	15500	3500
27	*18000*	*20500*	*5000*

Barks bands energy feature arrays are computed for three critical time locations:

- Beginning of element
- Q-time of element (3.3.2.7)
- End of element

Besides participating in computation of inter-elements distance these descriptors are also utilized for elements concatenation purposes to assess integration cost and guide continuity processing.

For magnitude spectrum M_t computed at time location t *Barks bands energy* is

$$E = \{B_1, B_2, \ldots, B_{27}\} \tag{3.3}$$

where B_i is energy at a specific bark band:

$$B_i = \sum_{j=min}^{max} M_{t_j}^2 \tag{3.4}$$

where M_{t_j} is magnitude of j_{th} spectrum bin and min and max are M_t's bin indexes defining given bark band frequency range.

3.3.3.4 Spectral RMS

Spectral RMS computes quadratic mean of bark bands:

$$S_{RMS} = \sqrt{\frac{1}{N} \sum_{i=1}^{N} B_i^2} \tag{3.5}$$

where B_i is a specific bark band's energy and N is the number of bark bands.
Alike *Barks bands energy* this descriptor is also computed for three time locations of element: beginning, q-time and end.

3.3.4 Tonal descriptors

Tonal descriptors contain two feature groups describing tonal characteristics of sound.

- Pitch group — represents pitch content
- Chroma group — depicts chroma generalizations of pitch

In following descriptions we will be using term *quota* to refer to percentage of the element's timespan during which some sound trait was observed. Commonly such value is expressed in normalized form in range $[0, 1]$.

Pitch descriptors group is based on pitch estimation analysis. Pitch estimation is computed as *instantaneous feature* [89], i.e. per each short-time frame over the course of sound element. Thus obtained estimated pitch sequence is further analyzed to extract pitch features grouped into three categories:

1. General category
    - Pitchiness
2. Strong pitch category
    - Pitch cardinality
    - Pitches
    - Pitch durations
    - Pitch quotas
    - Starting pitch
    - Ending pitch
3. Momentary pitch category
    - Momentary pitches cardinality
    - Momentary pitches
    - Momentary pitches duration
    - Momentary pitches quota

As can be seen from the above we define three pitch states: *pitch, momentary pitch* and *unpitched*. Two first states differentiate between prominent pitches and perceptually insignificant ones due to their only momentary appearance. This distinction helps to segregate pitches that carry out the element's tonal character from those that do not. Momentary pitches create tonal flickering rather than tonal character. For elements longer than 1 sec they are defined as pitched regions shorter than 100 ms, otherwise — as pitched regions occupying less than 20 percent of sound duration.

Chroma descriptors group is focused on describing harmonic and polyphonic properties of element, i.e. not only presence of specific pitch classes but also their simultaneity. This feature group is based on computation of HPCP — harmonic pitch class profile. The notion of pitch class profile was introduced in [48] where it is defined as "a twelve dimension vector which represents the intensities of the twelve semitone pitch classes". An extended approach to pitch class profile computation is proposed in [52] in which granularity is decreased to allow for finer precision than semitone and incorporates weighting and normalization into the processing.
HPCPs are computed as instantaneous vector and the resulting HPCP time series are used to extract following descriptors:

1. Chroma
    - Chroma cardinality
    - Chromas
    - Chroma durations
    - Chroma quotas
2. Polyphony
    - Chroma simultaneity ratio
    - Maximum chroma simultaneity

- Maximum chroma simultaneity local ratio

3.3.4.1 Pitchiness

This descriptor expresses general aspects of the sound's vertical regularity computed as three quota values:

- Strong pitches quota P_P
- Momentary pitches quota P_M
- Unpitched duration quota P_U

Together these three values add to 1:

$$P_P + P_M + P_U = 1 \tag{3.6}$$

This descriptor assists in quick assessment of the element's pitch prominence. The tripartite structure allows to assess distribution of time between pitched and unpitched durations as well as ratio between strong and momentary pitch spans.

3.3.4.2 Strong pitch descriptors

Strong pitches are those that last long enough to be perceptually salient, i.e. leave trace in the auditory memory. During the analysis user can modify temporal and quota thresholds that separate strong pitches from momentary ones. The default values are: 100 ms for elements longer than 1 sec and 20 percent for shorter ones.

Pitch cardinality describes number of strong pitches observed in element. This is the most general descriptor allowing quick interrogation into the pitch area of the element.

Pitches descriptor lists strong pitches detected during the course of element. If element contains multiple pitches they are listed in the order of prominence with the

longest sounding pitch coming first.

In sEl pitches are stored with one cent precision. They are encoded as real numbers with the whole part representing MIDI note number and fractional part representing cents. MIDI note numbers $\in [0, 127]$ covering frequency range $\in [8.18, 12544] Hz$. Middle C corresponds to the note number 60 and A4 (440 Hz) corresponds to note number 69. We use following formulae to convert frequency to pitch and vice versa:

$$p = 69 + \log_2(\frac{f - 440}{12}) \tag{3.7}$$

and correspondingly

$$f = 440 * 2^{\frac{p-69}{12}} \tag{3.8}$$

where f is frequency in Hz and p is pitch as *MIDI_note_number.cents* formatted value. As an example, detected frequency of 810 Hz will be stored as value of 79.565.

Pitch durations descriptor store each pitch's extent in seconds. As an addition to the above pitch list this array allows to inquire how long each pitch lasted.

Pitch quotas describe pitches durations in a relative way, i.e. how much each pitch contributed to the overall pitched duration. As usual quotas sum up to 1.

Starting and ending pitch descriptors specify enclosing pitches of an element

3.3.4.3 Momentary pitch descriptors

This group describes volatile pitch content that does not last long enough to become perceptually salient but nevertheless creates some statistical mass in tonal space.

Momentary pitches cardinality describes number of momentary pitches registered in the element.

Momentary pitches descriptor lists pitches in the order of duration with the longest momentary pitch listed first.

Momentary pitches duration is a single number indicating compound duration of momentary pitches in seconds.

Momentary pitches quota describes the extent of momentary pitches contribution to the overall pitched duration. If in formula 3.6 $P_P = 0.3$ and $P_M = 0.2$ then *momentary pitches quota* will be

$$Q_{P_M} = \frac{P_M}{P_P + P_M} = 0.4$$

3.3.4.4 Chroma descriptors

This group of descriptors characterizes chroma content of element.

Chroma cardinality stores number of chromas detected in element. This is the most general descriptor to assess the chroma content of an element.

Chromas descriptor lists chromas obtained via HPCP extraction during the course of element. In multiple chromas case they are ordered by duration with the longest chroma first.
In sEl chromas are stored as integers $\in [0, 11]$ indexing symbol array of twelve chromatic pitch classes as shown in table 3.25.

Chroma durations descriptor is an array with each chroma's span stored in seconds. Durations are listed in descending order which also corresponds to the order of chromas in the above descriptor.

Table 3.25: **Pitch classes symbols**

Index	Pitch class
0	A
1	A#
2	B
3	C
4	C#
5	D
6	D#
7	E
8	F
9	F#
10	G
11	G#

Chroma quotas describe each chroma's durational offering to the overall pitched time of the element. Chroma quotas sum up to 1.

3.3.4.5 Chroma polyphony descriptors

This descriptor group depicts aspects of chromas' simultaneous occurence.

Global chroma simultaneity ratio depicts contribution of polyphonic time to the overall duration of element:

$$S_G = \frac{D_S}{D_T}$$

where D_S is duration of simultaneous chromas and D_T is the total element's duration.

Local chroma simultaneity ratio stores relation between polyphonic time and pitched duration of element:

$$S_L = \frac{D_S}{D_P}$$

where D_P is the pitched duration.

Maximum chroma simultaneity shows maximum number of concurrent chromas observed in the element.

Maximum chroma simultaneity local ratio stores relation between time of *maximum chroma simultaneity* and overall simultaneity time:

$$R_{MaxS} = \frac{D_{MaxS}}{D_S}$$

where D_{MaxS} is duration of maximum chroma simultaneity.

3.3.5 Rhythm descriptors

In MIR applications rhythm-related descriptors are often utilized for tasks such as genre recognition ([41]), signal similarity assessment ([88], [47]) and others. Semantic access to rhythmic aspects of sound can be highly desirable for compositional decision-making as well. Information about layers of sub-audio periodicity present in the signal can be instrumental especially for *temporalization phase* of *NOTA-transform* pipeline (2.5.2.2).

A variety of methods has been proposed for rhythm feature extraction that run on a premise of dealing with inputs that actually contain rhythmic behavior. In an omni-source environment like sEl no such promise can be made. To extract rhythm-related descriptors from arbitrary signals we use general solution of *rhythm transform* described in [55]. The rhythm transform computes a number of frequency-band specific periodograms which altogether represent the rhythmic periodicities of a signal in rhythm domain similarly to how FFT depicts audio periodicities in frequency domain. Example of rhythm transform performed in sEl is demonstrated in figure 3.18: the plot shows four-five prominent peaks revealing polyvalent rhythmic structure. Each colored curve represents separate time frame in the rhythm domain.

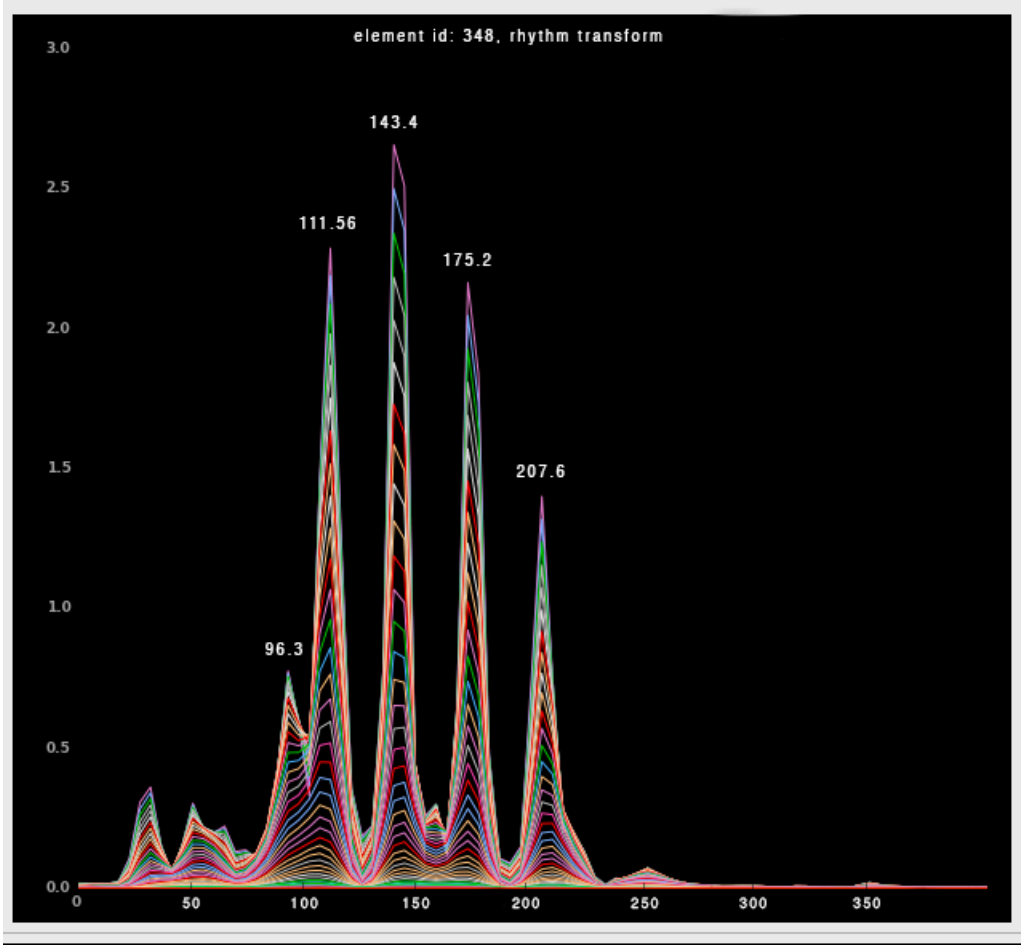


Figure 3.18: Rhythm transform plot in sEl.

Further analyzing rhythm transform of an element we extract the following features:

- CPMs
- CPM strengths
- CPM evolve
- Tatum

3.3.5.1 CPMs

Rhythm transform bins correspond to specific periodicity rates. Examining the bins we can find the energy of sub-audio periodicities found in the signal. From those

we pick energies rising above the minimum energy threshold and store corresponding periodicities as *CPMs* descriptor. Rhythm periodicities are stored in *cycles per minute* — a musically convenient unit similar to *bpm*. However we avoid using the term *beats* as having too narrow semantics for our purpose here.

To convert *rhythm domain bin index* to *cycles per minute* we use formula:

$$cpm_i = 30\frac{i \times f_s}{h \times w} \tag{3.9}$$

where i is a rhythm transfer bin index, f_s — a signal sample rate, h and w — hop size and frame size used to compute rhythm transform.

CPMs are stored in the decreasing order of prominence.

3.3.5.2 CPM strengths

Periodicities stored by the previous descriptor have energy levels associated with them. These levels are converted into relative strengths via division by the strongest level. The resulting coefficients are stored as *CPM strengths*. The order is the same as that of *CPMs*.

3.3.5.3 CPM evolve

Rhythmic periodicities may not stay fixed but evolve over the course of sound element. For each rhythm domain time frame we pick the strongest *CPM* and store it in a time series describing an evolution of the most prominent periodicity. *CPM evolve* is a dynamic descriptor for which *characteristic values* are computed (3.2.2.10).

3.3.5.4 Tatum

Tatum, a term coined in [20], refers to the high frequency pulse we often experience when in presence of music. Disabbreviated as *time quantum* this term "can be defined

as the lowest regular pulse train that a listener intuitively infers from the timing of perceived musical events" [63]. In sEl this descriptor is stored with some tweaks depending on the *timescale* (3.3.2.1) of a sound element:

- For elements that yield non-empty *CPMs* list the fastest rate is stored as tatum,
- For elements which are too short to contain rhythm cycles the very length of the element becomes a parameter to compute tatum. In this case *tatum* describes the rate of pulse that would result from the element's continuous repetition. This is a useful characteristic but nevertheless not a real tatum since the element itself does not have any. To mark such cases *tatum* is stored as a negative value.

3.3.6 Spectral descriptors

Spectral group includes low-level descriptors calculated from audio spectrum and characterizing various aspects of the sounds spectral shape. In sEl these features are predominantly used for element similarity searches. All descriptors in this group are dynamic thus characteristic attributes are computed to describe their temporal evolution (3.2.2.10). The spectral group comprises following descriptors:

- Spectral complexity
- Spectral centroid
- Spectral decrease
- Spectral flux
- Spectral peakiness
- Spectral rolloff
- Spectral spread
- Spectral skewness
- High frequency content

3.3.6.1 Spectral centroid

This descriptor represents center of spectral mass which correlates with perception of the sound's brightness or dullness. It is computed as a mean value of frequency distribution across the magnitude spectrum where magnitude values are used as frequency weights:

$$S_\mu = \frac{\sum_{k=1}^{N} m[k] f_k}{\sum_{k=1}^{N} m[k]} \tag{3.10}$$

where N is number of spectrum bins, k — bin index, $m[k]$ — magnitude and f_k — k_{th} bin frequency. Spectral centroid is represented in Hz units.

3.3.6.2 Spectral complexity

This feature describes the complexity of a signal frame in terms of multiplicity of frequency constituents. It is defined in [68] as the count of the spectral peaks. To compute this descriptor we apply peak detection algorithm to the magnitude spectrum and count peaks that surpass magnitude threshold — a user controllable parameter.

3.3.6.3 Spectral decrease

Taking spectral shape as a linear regression model this descriptor is a least squares regression estimator that shows the linear degree of spectral decrease (or increase).

3.3.6.4 Spectral flux

This descriptor represents temporal variability of the magnitude spectrum on a frame-to-frame basis. It can be computed as an X-norm difference between two consecutive spectral frames. In some applications X=1 [40]:

$$S_f = \sum_{i=1}^{N} |m_i - m_{i-1}| \tag{3.11}$$

while in others X=2 [116], [86]:

$$S_f = \sqrt{\sum_{i=1}^{N} (m_i - m_{i-1})^2} \tag{3.12}$$

where m_i and m_{i-1} are magnitudes of current and preceding frames. In sEl $X \in [1, 2]$ is a user selectable parameter.

3.3.6.5 Spectral peakiness

Spectral peakiness (also called kurtosis) identifies the degree of flatness of the spectral shape around the centroid. Lesser value accounts for smoother shape. This descriptor is computed as fourth order central moment normalized by sum of magnitudes:

$$S_\delta = \sqrt[4]{\frac{\sum_{k=1}^{N} m[k](f_k - S_\mu)^4}{\sum_{k=1}^{N} m[k]}} \tag{3.13}$$

3.3.6.6 Spectral rolloff

Spectral rolloff is a frequency below which the cumulative energy of the signal reaches specific percent k of the total energy (usually $k \in [85, 90]\%$):

$$\sum_{0}^{f_r} m_f^2 = k \sum_{0}^{f_{Ny}} m_f^2 \tag{3.14}$$

where f_r is rolloff frequency, f_{Ny} — Nyquist frequency and m_f — magnitude of frequency f within the spectrum.

This descriptor is measured in Hz, it delineates frequency range containing most of the signal's energy.

3.3.6.7 Spectral skewness

This descriptor outlines spectral shape's asymmetry with respect to the *spectral centroid*. Zero skewness corresponds to symmetric spectral shape, negative value indicates prevalence of higher frequencies and positive value — the opposite. *Spectral skewness* corresponds to the third central moment of frequency distribution:

$$S_\gamma = \sqrt[3]{\frac{\sum_{k=1}^{N} m[k](f_k - S_\mu)^3}{\sum_{k=1}^{N} m[k]}} \tag{3.15}$$

3.3.6.8 Spectral spread

Spectral spread characterizes concentration of spectral energy around the spectral centroid and thereby delineates its global bandwidth. Spectral spread corresponds to the spectrum's standard deviation normalized over the sum of the magnitudes:

$$S_\sigma = \sqrt{\frac{\sum_{k=1}^{N} m[k](f_k - S_\mu)^2}{\sum_{k=1}^{N} m[k]}} \tag{3.16}$$

3.3.6.9 High frequency content (HFC)

High frequency content sums up spectrum magnitudes while assigning progressively more weight to higher frequencies. This descriptor's sensitivity is honed towards the higher end of the spectrum. Different applications differently boost high frequencies weights during HFC computation. In [25] HFC is essentially a spectral centroid without normaliztaion:

$$hfc = \sum_{0}^{f_{Ny}} f m_f \tag{3.17}$$

while in [72] magnitudes are squared:

$$hfc = \sum_{0}^{f_{Ny}} f m_f^2 \tag{3.18}$$

and in [64] squared are the frequencies:

$$hfc = \sum_{0}^{f_{Ny}} f^2 m_f \tag{3.19}$$

In sEl either option is available.

3.3.7 Integrated architecture

sEl's *analysis subsystem* is implemented as integrated architecture that can take advantage of functionality provided by external third-party music information retrieval toolsets. External functionality is added to sEl's analysis subsystem via implementing integration layer that adapts each toolset's API to the internal analysis protocol.

3.3.7.1 Integration layer

Each external toolset that sEl embeds is communicated via dedicated integration object (called *engine*) which translates calls between the system and a toolset. Every toolset has its own API for setting parameters and formatting analysis input. Besides resolving these, integration objects also publish their toolsets' feature computing capabilities so that sEl can compile a system-wide library of computable features.
Normally features are computed as a chain of interconnected lower-level computations compoundly called *analysis graph*. Due to the integration abstraction mechanism different toolsets can compute different parts of the same analysis graph without knowing about each other. A general scheme of toolsets integration is shown in figure 3.19.

3.3.7.2 External toolsets

Two last decades of research in musical informatics brought to life numerous software packages to assist in music information retrieval tasks. Varying in capabilities, platform and licensing those toolsets form the computational resource base for

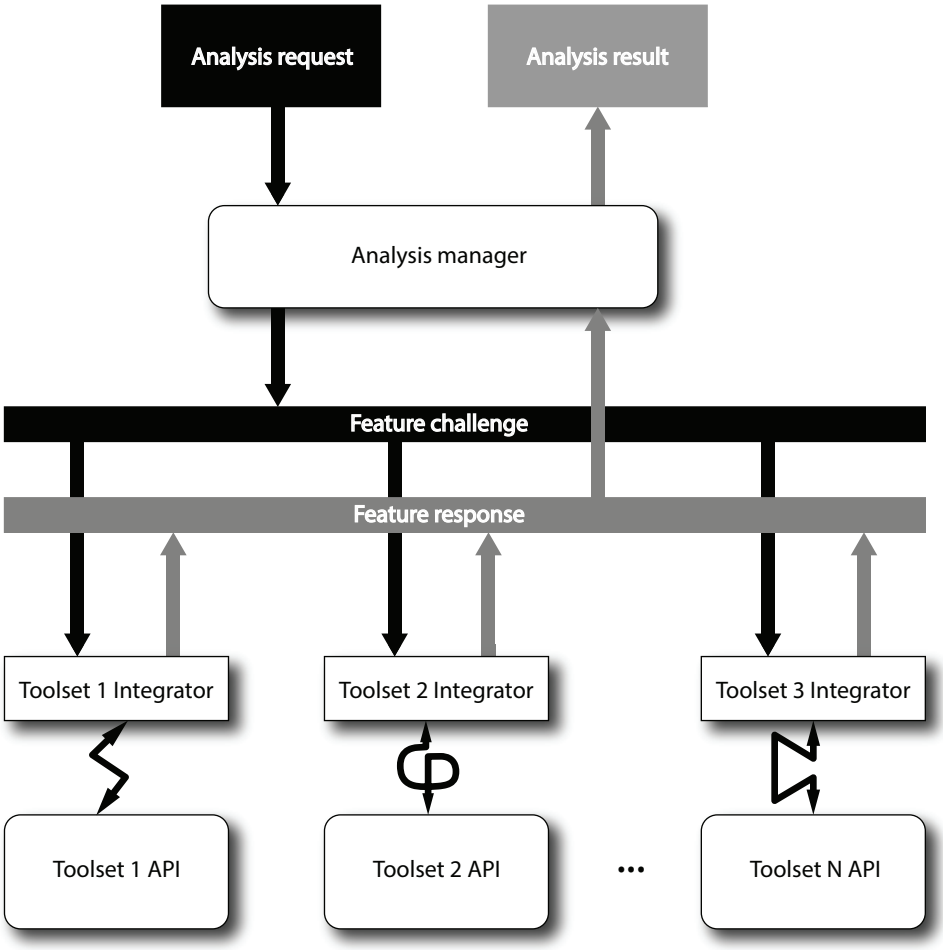


Figure 3.19: External toolsets integration in sEl.

researchers.

For a long time the natural platform choice for MIR toolsets has been Matlab, with its wealth of tools for signal processing and multi-dimensional arrays manipulation. Besides narrow-task MIR libraries, some general toolsets have been developed, such as *Timbre toolbox* [90], *MIRtoolbox* [67], *EASY* [87] to name a few. A number of general MIR toolsets have been implemented using Java language such as *jAudio* [79] or *AMUSE* [119]. Third widespread language platform for developing MIR libraries is C/C++ which was used to implement such commendable toolsets as *aubio* [25], *Essentia* [22], *LibXtract* [26] and *Marsyas* [117], [115]. Some toolsets are

implemented as web service, e.g. *Echonest* [44] or *jWebMiner* [80].

During research into external toolsets to integrate with sEl we looked for those that could be interfaced with Python. Some toolsets, like *Bregman toolbox* [32] and *YAAFE* [74] were developed in Python which makes them obvious candidates for integration. The aforementioned C/C++ toolsets (aubio, Essentia, LibXtract and Marsyas) provide Python front-ends which also makes them directly accessibly from sEl. In addition to this selection Echonest supplies Python API to connect to their server. Appendix C provides a categorized comparison between capabilities of these seven MIR toolsets.

Of all the candidate libraries, the easiest to integrate turned out to be *aubio* and *Essentia* due to their modular architecture and seamless support of *NumPy*[10] arrays. Currently these two libraries are integrated into sEl and support for the rest is a work for the future.

3.3.8 Flexibility

Analysis subsystem of sEl offers control over the actual set of features that will be computed for new elements added to the *material layer* (3.2.1) of the database. By manipulating *active* field of *dbDescriptor* (3.2.2.1) database object features can be turned on and off.

3.3.9 Visualization

Initially sEl printed analysis data to the console but soon enough it became clear that the amount of information exceeds any convenience thresholds with the chosen

[10] *NumPy* is Python library for working with multi-dimensional arrays. Backed by fast C-compiled core, with its functionality and performance rivaling that of Matlab, *NumPy* became a de facto standard for array and matrix manipulation in Python.

method of visual feedback. This necessitated implementation of data visualization module to make working with elements analysis more efficient and enjoyable. The current implementation is based on the functionality of *matplotlib*[11] library [85] embedded into sEl's GUI. The module, which is still far from being complete, allows to visualize analysis data for the user. In the future visualization module will be redesigned towards integrated architecture with the ability to uniformly interface other external visualization packages, especially OpenGL based, like *Galry* [98] or *VISPY* [29]. Figures 3.20 and 3.21 show visualization examples for *pitch map* analysis and *HPCP* descriptor.

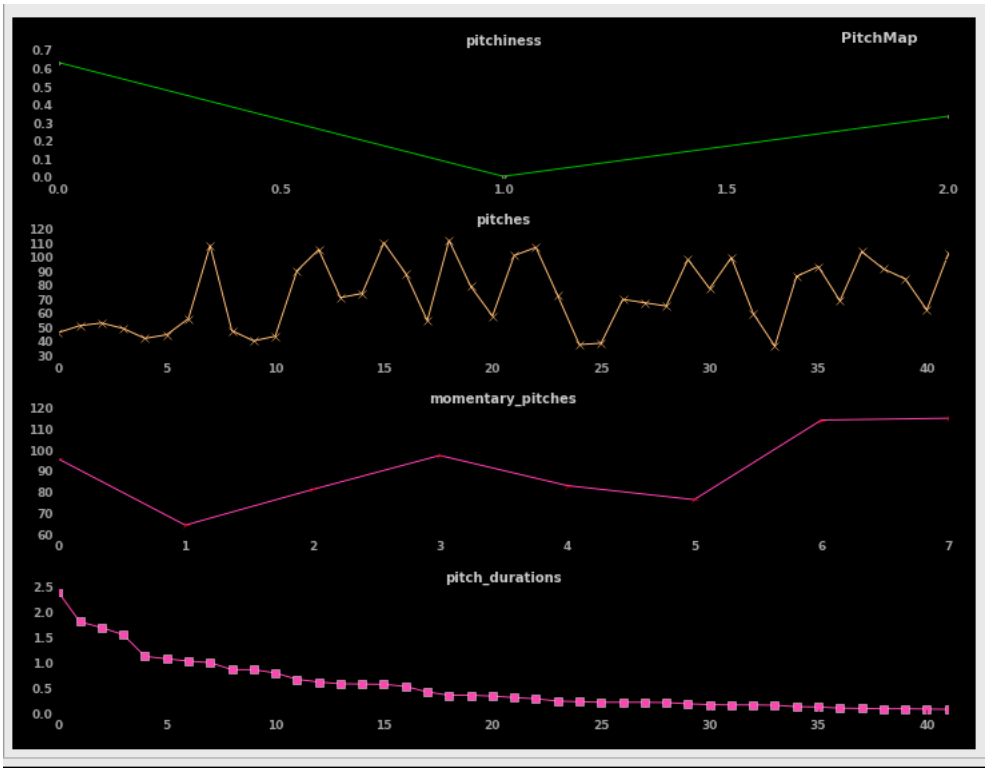


Figure 3.20: Pitch map analysis visualization.

[11]Matplotlib is data plotting library designed to provide functionality similar to that of Matlab.

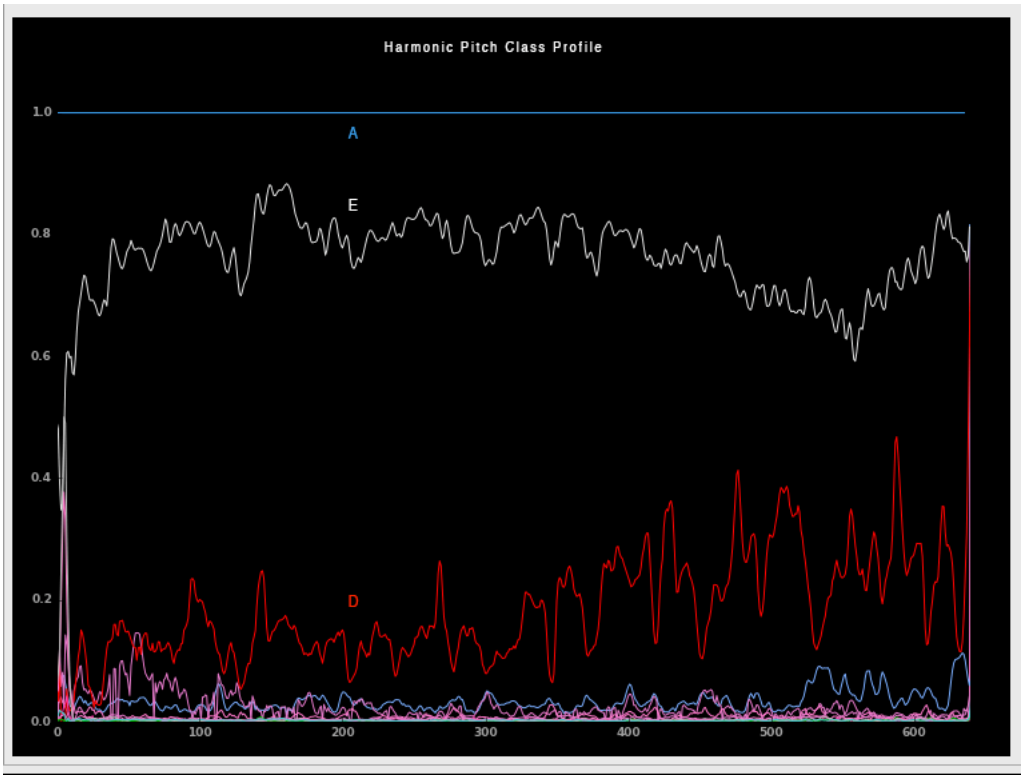


Figure 3.21: Harmonic Pitch Class Profile (HPCP) visualization.

3.3.10 Extensibility

sEl's *analysis subsystem* is designed to encourage experimentation and facilitate extensions of the library of analysis elements. In addition to running analyses and saving them to database user can experiment with different configurations of analysis pipelines by putting together available processing elements and modifying their parameters. Processing construct that takes audio signal as input and produces one or more extracted features is called *analysis graph*.

3.3.10.1 Analysis graph

Typically feature extraction process consists of a number of steps where each step takes input data from a previous step, transforms data in a particular way and passes it further. For instance, to compute *HPCP* of a signal we need to

1. Remove DC offset from the signal,

2. Apply equal loudness filter to better align the result with perceptual space,
3. Split the signal into overlapping short time frames,
4. Multiply each frame by specific window function,
5. Perform FFT on windowed frame,
6. Extract spectral peaks from magnitude spectrum of the FFT,
7. Extract chroma array energies from the spectral peaks.

In sEl's parlance the overall analysis sequence is called *analysis graph* and each step in this process is called *analysis node*. Analysis node's structure involves

- a processing unit
- input(s)
- parameters
- output(s)

Figure 3.22 shows the above analysis realized a processing nodes' graph. This particular calculation does not contain execution branching but other processes do. The figure also shows that *analysis graph* allows *processing nodes* to communicate on parameter level to ensure consistency of parameters with identical semantics like *sample rate* or *frame size*.

When various features are being computed for a sound element, it is certain that the same processing nodes will be used multiple times. For instance, the magnitude spectrum is used as input for all *spectral descriptors* (3.3.6), *bark bands* (3.3.3.3), *chroma descriptors* (3.3.4.4) and more. To avoid unnecessary wasting of memory and processing time due to duplication, sEl maintains internal caching of *processing nodes* outcomes and returns the cached data whenever such a node is called for processing. To put it differently, any node computes its data only once for a given input and while having the same parameters settings. However, when parameters settings differ

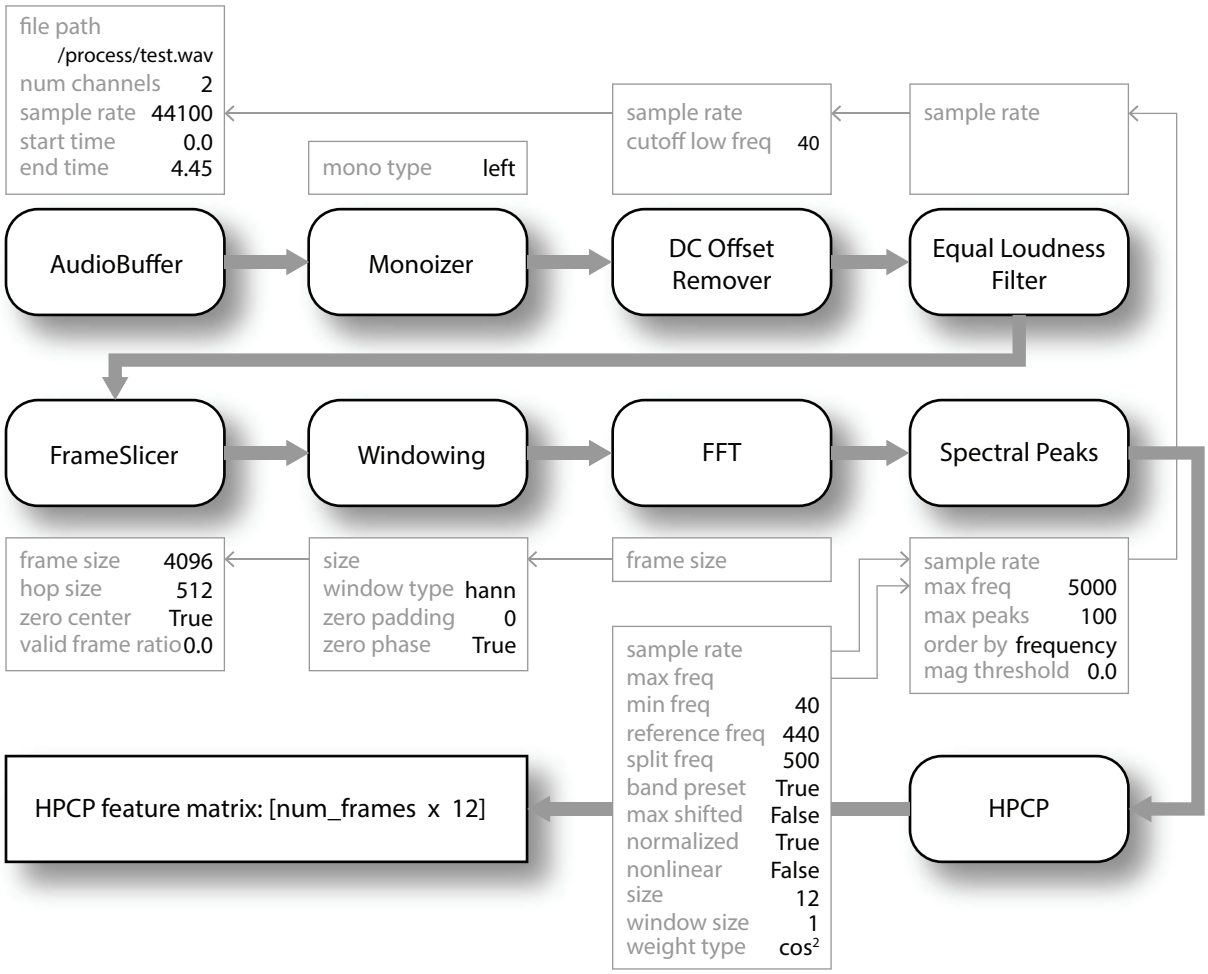


Figure 3.22: Example of analysis graph.

even slightly, the result is computed anew (as it should). New computation does not destroy any cached results because they may be called for by other nodes. Caching optimization works automatically without requiring any control from the user.

3.3.10.2 Extending the analysis library

sEl comes with a set of pre-assembled analysis graphs sufficient to compute the default features (3.3.1). For further experimentation and extension the software provides different ways to do so.

Defining new analysis graphs can be done via *GraphManager* object which can be used to assemble arbitrary graphs out of the available *processing nodes*. Currently this can only be done by writing Python code. In the future for users who do not program an audio graph editor will be implemented to allow assembling analysis graphs by methods of visual programming (similar to *PureData* [95], *OpenMusic* [10] and the like).

Adding new processing nodes is another method of extensibility in sEl's analysis subsystem. New processing nodes can be written by subclassing *ProcessingNode* base class and implementing few methods that describe new node's parameters as well as writing *process* method that does actual processing. This method will be called by the analysis framework and passed the necessary input data for computation. Despite being part of an analysis graph any node in its *process* method can internally assemble its own working graphs out of other nodes to assist in computation.
After new processing node classes are implemented no additional effort is required to make them part of the analysis library: *GraphManager* will collect all processing node classes from the code automagically during the sEl's start sequence so that they will be available as building blocks for assembling analysis graphs.

For experimentation with analysis graphs sEl provides GUI interface which allows to select specific graph, run it on sound elements and observe results in both console window and visualization widget (3.3.9).

3.3.11 Element features assessment

Each database element's computed features can be assessed via activating *Analysis window*[12] shown in figure 3.24. For easier evaluation descriptors are grouped by semantics and color-coded.

3.4 Compositional subsystem

The *NOTA-transform* described in (2.5.2.2) is an essential scheme of recombinant usage of EncycloSpace in compositional context. Existence of semantic layer of the EncycloSpace is a prerequisite for that process. This prerequisite is addressed by *Analysis subsystem* which elevates the opaque material level of the EncycloSpace (audio) to become a navigable semantic space of features. However descriptive this space might be it does not contain any compositional structures. This is the task of *Compositional subsystem*.

Implementation of *Compositional subsystem* complies with the tripartite structure of NOTA-transform in offering tools for each of its transformational phases, i.e. *Navigation, Ordering* and *Temporalization*.

3.4.1 Navigation

As detailed in 2.5.2.2 *Navigation* is the first phase of the recombinant process. It creates projection of EncycloSpace, called N-projection (Eq. 2.2), containing elements that satisfy a set of constraints called *navigation context* (2.5.2.3). sEl allows to create navigation contexts of arbitrary complexity via Python classes *dbSearchContext* and *SearchClause*. A single *dbSearchContext* object contains collection of *SearchClause* objects each of which defines some aspect of navigation. These aspects are applied

[12] Analysis window shows static features data and mean values for dynamic features. Characteristic data for dynamic features is not displayed. GUI element for assessing characteristic values of dynamic features will be added in the future versions of sEl.

collectively during the database search.

SearchClause is an abstract class which defines interface for navigation elements. sEl provides two classes derived from *SearchClause* to assist in specific types of navigation:

- *FeatureSearchClause* selects sound elements with specific feature having a certain value range or participating in a specific category.
- *ChildrenSearchClause* selects sound elements on the basis of containment within other elements.

Once navigation context is established it can be passed to the sEl's navigation API which will administer database search and return N-projection — a specific collection of elements which satisfied the navigation context. The returned collection can be passed to the next *NOTA-transform* phase or saved to the database for later use as a new *dbCollection* (3.2.3.1) of type *set*.

3.4.1.1 Navigation extensibility

Using *SearchClause* interface user of the system can expand navigation vocabulary by deriving new navigation classes.

3.4.2 Ordering

Element collections obtained via navigation can be used to construct music entities of very different time scales, the actual scale depending on the collection's content and the compositional strategy. In any case there is a distinct decision-making stage defining which elements and in which order should the music object contain. This process corresponds to the second phase of *NOTA-transform* called *Ordering*. As shown in Eq. 2.4 *ordering* takes as input N-projection and outputs O-projection — an ordered sequence of sound elements. This transformation requires an *ordering context* (2.5.2.4) — a mechanism capable of converting unordered set of elements into

ordered sequences.

sEl offers an ordering interface defined in *ElementOrderBase* class. sEl comes with a number of ordering classes derived from *ElementOrderBase*:

- *FeatureOrder* — creates sequence of elements based on values of specific features,
- *RandomOrder* — creates randomized sequences,
- *NaturalOrder* — preserves whatever order the collection already has.

Orders can also be cascaded to create more sophisticated O-projections. Ordered elements can be passed to the last *NOTA-transform* phase — temporalization but they can also be save in the database for later use as a *dbCollection* (3.2.3.1) of type *sequence*.

3.4.2.1 Ordering extensibility

The roster of ordering classes can be expanded: by deriving from *ElementOrderBase* interface user can implement arbitrary element ordering strategies: deterministic and non deterministic, using various mappings and probability distributions, but most interestingly — strategies driven by expanding and utilizing sound descriptors semantics.

3.4.3 Temporalization

Temporalization is the third phase of the *NOTA-transform* which allows recombination of elements on a temporal scale. This phase is defined in Eq. 2.6 as conversion from O-projection to T-projection via usage of *temporal context* (2.5.2.5) — a collection of time structures that can be used to distribute elements along the timeline and modify their temporal span.

sEl supports *temporalization* via Python classes *TemporalContext* and *TimeStructure*.

Each composition object in sEl contains *TemporalContext* which in turn can contain a number of *TimeStructures*. Each *TimeStructure* can temporailze a single *strand* (2.5.2.1) — a sequential line of compositional *events*. Once *event* is created within the *strand* which has a *TimeStructure* associated with it, the event's content (sound elements) can be precisely aligned against the points of the *TimeStructure*.
Time structures can be created in a couple of different ways:

- As a sequence of absolute time points,
- As a symbolic time structure — a sequence of time signatures and tempi

Both approaches can be used either deterministically by direct specification or algorithmically via arbitrary means of computation.
Time points can be assigned structural roles which allow precise control over the time alignment. Currently supported roles are:

- Measure
- Beat
- Sub-beat
- Tatum
- Time point

Each element that can be a subject of temporalization can be given a list of roles which it will respond to. For instance, elements that respond strictly to *Measure* role can only be controlled by time points which have this role assigned.
The list of roles can be expanded by user to include arbitrary structural semantics.

3.4.3.1 Absolute time

Absolute time points specify time structure in terms of physical time units. This method is favorable for rubato like time structuring.

3.4.3.2 Symbolic time

Symbolic time allows to use the language of musical time signatures and tempi to specify how physical time points should be unwrapped. This syntax is beneficial for well-structured temporal definitions.

3.4.3.3 Hybrid time

Both absolute and symbolic time notations can be freely mixed. In that mode absolute time points represent the time elapsed since the beginning of the given absolute time section.

3.4.4 Adaptation

Adaptation is the concluding stage of *NOTA-transform* which takes care of final adjustments of sound elements in order to produce an expected sound result. In sEl *adaptation* is realized as a feature-guided ability of compositional objects to modify their content during the rendering. For instance, a sound element, given the reference of another sound element, can adjust its loudness, pitch and duration accordingly, administer edge-fading etc.

3.4.5 Compositional objects

Four types of *compositional hierarchy objects* described in 2.5.2.1 are implemented in sEl as Python classes *Composition, Section, PerformanceStrand* and *PerformanceEvent*.

3.4.5.1 Composition class

Composition class represents the whole music work and holds its metadata (title, composer etc) and a list of sections.

3.4.5.2 Section class

Section class holds a list of section strands and a temporal context. Strands represent parallel layers of sound. Temporal context contains time structures designed for the section.

3.4.5.3 Strand class

Strand class consists of events — temporal containers that have time position and duration. Strand can be assigned a time structure from the section's temporal context in which case the strand's content will be temporally controlled by that structure's time points.

3.4.5.4 Event class

Event class represents actual sound content for which it defines temporal bounds and position. The content can be any object derived from the class *PerformanceObject* which defines interface for any playable sound content. Currently there are three playable classes — *SoundElement*, *ElementCollection* and *MetaCollection*. *SoundElement* represents a single material entry of EncycloSpace — audio element stored in *dbElement* table (3.2.1.3). *ElementCollection* and *MetaCollection* embody N-projections and O-projections of EncycloSpace stored in *dbCollection* (3.2.3.1) or *dbCollectionGroup* (3.2.3.4) tables correspondingly. For the current version of sEl it was crucial to implement these three content types but in the future versions other sound production classes can be added.

Since single *Event* can contain whole collections of elements this opens up the potentiality for different strategies of usage. Moreover, several *Events* can refer to the same *ElementCollection* while occupying different time locations and having different duration spans. Via sEl's *ordering* and *temporalization* API such *ElementCollection* can be deployed, for instance, with or without re-triggering of collection, with or without

duplication of elements etc. Overall recombinant possibilities are too numerous to be recounted here.

Figure 3.25 shows relationships between sEl's compositional objects.

3.5 Rendering subsystem

Since the *Composition* object is only a symbolic specification of the composition, *rendering* is required to obtain the final result (2.5.3). The rendering subsystem takes the composition object, parses it, instantiates required performance objects and creates a perceivable result. The rendering functionality is encapsulated in the Python class called *Performance*. During the class construction, rendering parameters can be passed to the constructor such as sample rate, number of output channels etc. *Performance* object can also accept starting time position and duration to allow selective rendering of composition's segments.
Current implementation supports rendering result as a waveform. Future versions will also implement symbolic rendering mode. This will allow to produce specific parts of the composition as symbolic notation to facilitate inclusion of live instrumentalists into the performance.

Rendering starts with a *pre-rendering phase* whose objective is to resolve indeterminacy contained in compositional structures. Indeterminacy can exists due to probabilistic constructs created during compositional decision-making. After pre-rendering the compositional model becomes wholly deterministic and thus renderable. However, even then rendering might not be completely linear due to possible *element nudging* during the mixing process. Element nudging has to do with temporal tightening based on the usage of temporal descriptors such as *Q-time*.

Paragraph 3.3.2.7 explains this in detail.

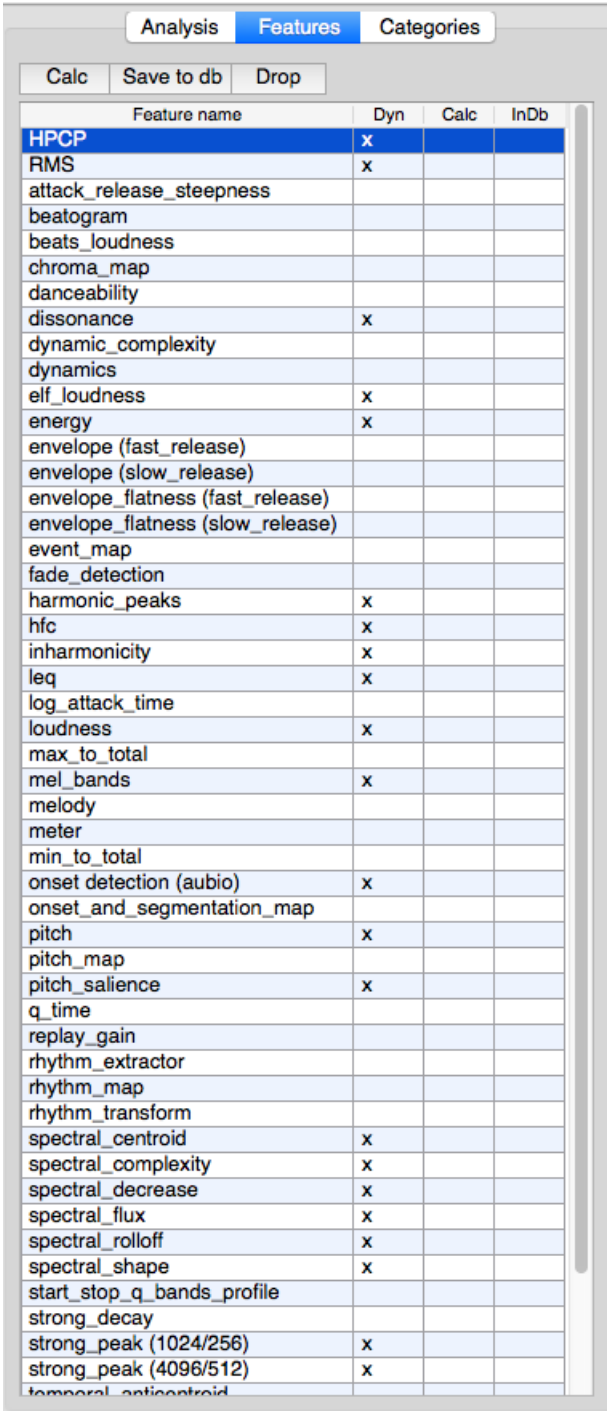
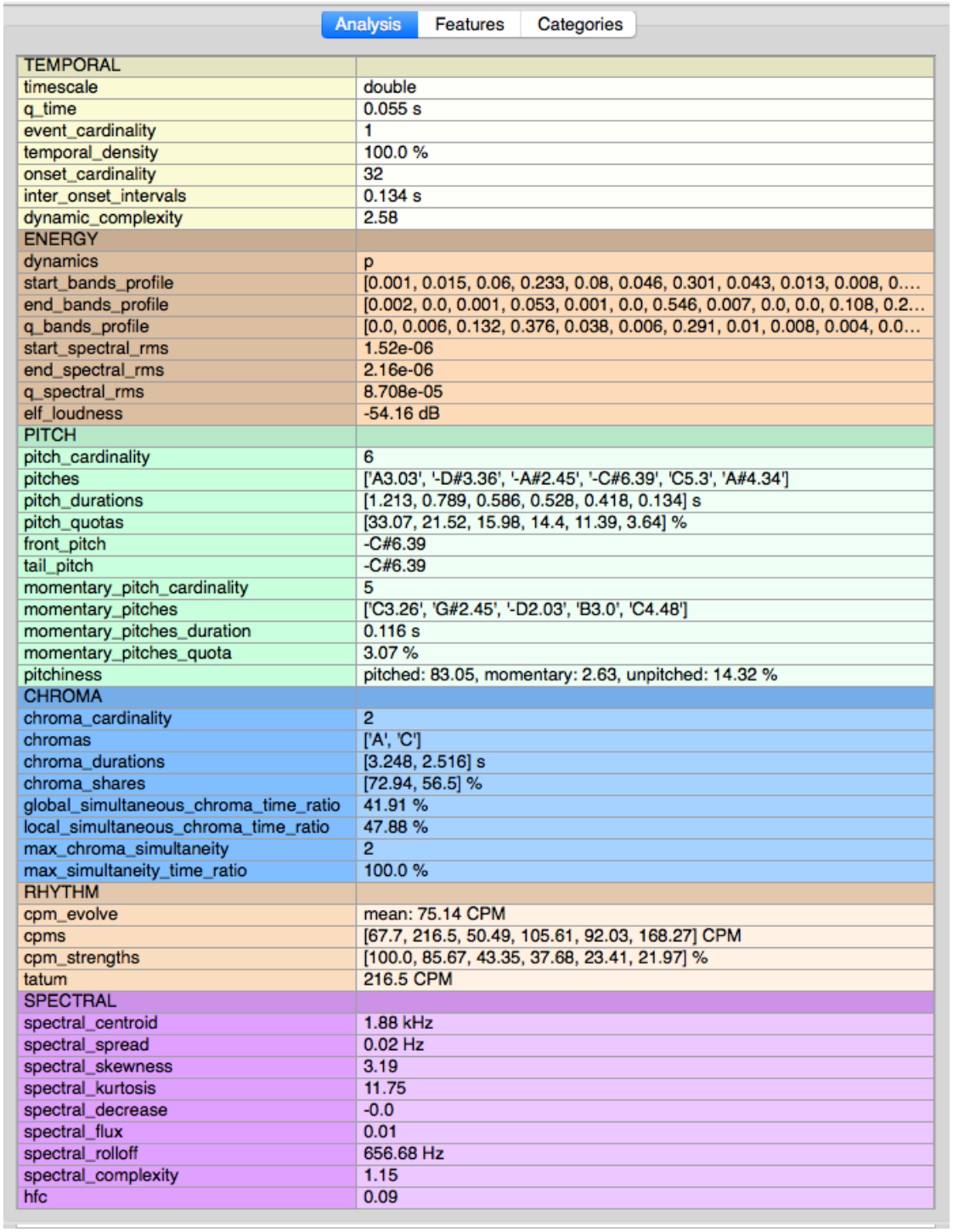


Figure 3.23: Feature analysis graph selection widget



TEMPORAL	
timescale	double
q_time	0.055 s
event_cardinality	1
temporal_density	100.0 %
onset_cardinality	32
inter_onset_intervals	0.134 s
dynamic_complexity	2.58
ENERGY	
dynamics	p
start_bands_profile	[0.001, 0.015, 0.06, 0.233, 0.08, 0.046, 0.301, 0.043, 0.013, 0.008, 0....
end_bands_profile	[0.002, 0.0, 0.001, 0.053, 0.001, 0.0, 0.546, 0.007, 0.0, 0.0, 0.108, 0.2...
q_bands_profile	[0.0, 0.006, 0.132, 0.376, 0.038, 0.006, 0.291, 0.01, 0.008, 0.004, 0.0...
start_spectral_rms	1.52e-06
end_spectral_rms	2.16e-06
q_spectral_rms	8.708e-05
elf_loudness	-54.16 dB
PITCH	
pitch_cardinality	6
pitches	['A3.03', '-D#3.36', '-A#2.45', '-C#6.39', 'C5.3', 'A#4.34']
pitch_durations	[1.213, 0.789, 0.586, 0.528, 0.418, 0.134] s
pitch_quotas	[33.07, 21.52, 15.98, 14.4, 11.39, 3.64] %
front_pitch	-C#6.39
tail_pitch	-C#6.39
momentary_pitch_cardinality	5
momentary_pitches	['C3.26', 'G#2.45', '-D2.03', 'B3.0', 'C4.48']
momentary_pitches_duration	0.116 s
momentary_pitches_quota	3.07 %
pitchiness	pitched: 83.05, momentary: 2.63, unpitched: 14.32 %
CHROMA	
chroma_cardinality	2
chromas	['A', 'C']
chroma_durations	[3.248, 2.516] s
chroma_shares	[72.94, 56.5] %
global_simultaneous_chroma_time_ratio	41.91 %
local_simultaneous_chroma_time_ratio	47.88 %
max_chroma_simultaneity	2
max_simultaneity_time_ratio	100.0 %
RHYTHM	
cpm_evolve	mean: 75.14 CPM
cpms	[67.7, 216.5, 50.49, 105.61, 92.03, 168.27] CPM
cpm_strengths	[100.0, 85.67, 43.35, 37.68, 23.41, 21.97] %
tatum	216.5 CPM
SPECTRAL	
spectral_centroid	1.88 kHz
spectral_spread	0.02 Hz
spectral_skewness	3.19
spectral_kurtosis	11.75
spectral_decrease	-0.0
spectral_flux	0.01
spectral_rolloff	656.68 Hz
spectral_complexity	1.15
hfc	0.09

Figure 3.24: Element features window

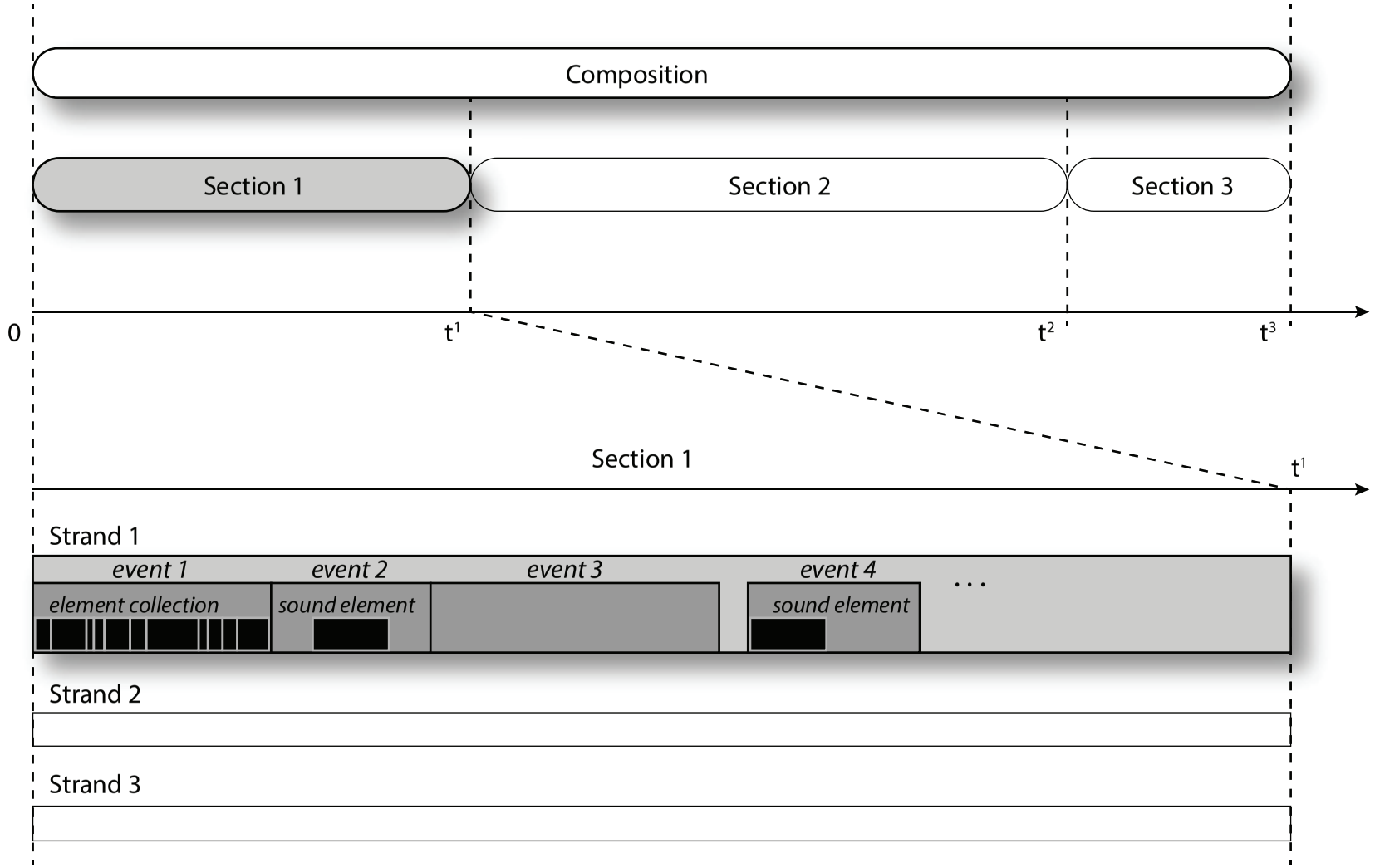


Figure 3.25: Structural relationships between compositional objects

Chapter 4

Ignis Fatuus: a recombinant piece composed with sEl

This chapter describes *Ignis Fatuus*, a piece composed using sEl. Our objective was to showcase compositional affordances of the software and to test its functional performance in practice. The composition is an acousmatic work, that comprises six movements, each employing particular strategy based on *NOTA-transform*.
The piece's title reflects our intention of constructing illusory sonic identities based on recombinant constructs of natural sounds.

4.1 The composition's EncycloSpace

The material layer (3.2.1) of the piece's EncycloSpace consists of 1,291 sound files. The files comprise two types of material:

- Sampled sounds — personal recordings of voice, piano and variety of objects made of glass, plastic, rubber, metal, stone etc
- Synthesized sounds — waveforms, algorithmically generated by a personal algorithm called *SpecGen* created in the RTcmix environment ([51]).

The semantics layer (3.2.2) comprises 18,394 elements from which 764,403 descriptors were extracted by the *analysis subsystem* (3.3).

The recombinant layer (3.2.3) contains 133 collections, created in sEl both by utilizing user interface and by executing *navigation contexts* (3.4.1) from compositional scripts.

4.2 Movement I. *Have I destroyed? (Intro)*

The phrase "Have I destroyed?" is both a key semantic motif for the piece and starting words of a poem by an American poet Jeremy Ward who recited it for my recording back in 2005. In this opening movement the first line of the poet's recital is heard untouched, after which it is algorithmically recombined using the height of the voice tone as an ordering principle and the loudness threshold as a navigation criterion to filter out segments representing pauses in the speech, breath and miscellaneous noises. As a result the movement builds up to a culmination using natural characteristics of the reader's prosody.

NOTA-transform-related facts:

- *Navigation* is a two-step process:
    1. selection of elements based on file membership
    2. secondary selection based on the loudness threshold
- *Ordering* is based on increasing values of pitch and loudness combined.
- *Temporalization* is *lazy*, i.e. it uses time positions and durations produced as a result of ordering.
- *Adaptation* includes 5 msec cross-fading between elements.

This movement was completely produced within sEl.

4.3 Movement II. *Vocalise*

This movement's sound material is based on sound recordings of my voice: abstract articulations, lip noises, breathing, glottal sounds etc. Some of the recordings were processed by the *SpecGen* script before being placed in the database. A compositional structure of the movement utilizes 6 collections subsequently distributed among 4 *strands* (3.4.5.3): a main voice strand, a low-register strand and two arrangement strands.

The primary ordering principle, employed in *Vocalise* is *automapping* — a collection orders itself by starting from some element (chosen deterministically or stochastically) and then replacing that element with another one, found via computation of the descriptor-based distance. The selected (found) element replaces the first one and vacates its own index in the sequence. To fill that index, another search takes place, an so on until no elements are left to choose from. The last place is taken by the element, from which the mapping has started. This provides closure for the mapping procedure.

For the main strand's temporalization, time points were produced by the ordering stage, other strands derived their time structures from the main strand's segmental locations. Unlike the arrangement strands, the low-register strand's collection was ordered by method, called *recombinant variation*, which allows multiple instantiation of elements and produces order variations which are seamless non-repeating and texture-like. The final assembly of the strands was done using an external DAW[1], because the compositional subsystem does not yet have the necessary interface.

[1]Digital Audio Workstation

4.4 Movement III. *Upbound*

This movement is composed from synthetic material generated by the *SpecGen* algorithm mentioned above. The compositional method is based on the intention to build an artificial articulation stream imitating dynamics of the prosodic intonation with rises and falls, intermittent rhythmicity, breathing and mixing the noisy and pitched components. In parallel, the music traverses the registral space in an unnoticeably slow gradual fashion so, when the movement ends, the listener finds herself at an opposite end of the sound spectrum from that of the movement's start.

To achieve this goal we created two collections from the *SpecGen* material using navigation contexts based on the *pitchiness* (3.3.4.1) descriptor to separate noisy and pitched elements. Both collections were individually ordered by using the following multi-step process:

1. An ordering based on the *strongest pitch* (3.3.4.2) (for the pitched collection) or *spectral rolloff* (3.3.6.6) (for the non-pitched collection) was applied first.
2. In a subsequent ordering the elements were rearranged based on their durations to imitate prosodic irregular rhythmicity

During the *temporalization* stage the pitched collection, as smaller of the two, was matched against the unpitched one in order to eliminate the temporal difference by spreading the shorter collection's elements along the larger collection's duration. At the *adaptation* phase, besides the usual 5-msec cross-fading, each element's volume was scaled to match -10 dB. For calculation the element's *ELF loudness* descriptor (3.3.3.2) was used. This movement was mostly produced in sEl with the exception of the final assembly of strands.

4.5 Movement IV. *Toccata*

This movement was composed to showcase temporal synchronization capabilities of sEl. It comprises 18 collections to build up the timbral space. These collections were assembled by navigation based on file membership and, most critically, on the *timescale* descriptor (3.3.2.1) to return elements shorter than 50 msec. Each collection was then separately ordered: depending on the timbral properties, either pitch-based or random ordering was used. Consequently, the collections were temporalized by *time structures* (3.4.3) based on *prestissimo* tempo (200 bpm). Each element's placement was guided by its *q-time* descriptor (3.3.2.7) for precise temporal positioning. The final assembly of the generated strands was administered in an external DAW.

4.6 Movement V. *Ignis Fatuus*

This penultimate movement is completely composed out of sounds produced by a set of glass objects, shown in figure 4.1. This set was purposely assembled for the composition of this movement. By navigation, out of about 6 hours of recordings, 22 collections were produced based on the criterion of performing technique. Further, sEl was used to print the element-wise pitch information from which the compositional pitch-plan was constructed. Due to the complexity of the sound material, the rest of this movement's composing was done manually.

4.7 Movement VI. *Have I destroyed? (Coda)*

The concluding movement reiterates the material from the beginning of the piece — that of the poem recital by Jeremy Ward. A different strategy is used, however: the narrator's voice struggles to be heard through three consecutively emerging recom-

Figure 4.1: A set of glass objects used to record the sound material for *Ignis Fatuus*

binant versions of itself produced by the *music mosaicing* method (2.5.2). Three different collections were used to produce mosaicing:

1. Synthetic sounds by *SpecGen* algorithm
2. Recordings of balloon rubbing
3. Recordings of a knitting needle vibrating.

Along with paying tribute to *music mosaicing*, this movement also demonstrates one of the possibilities for a recombinant use of this synthesis method. The movement was completely produced within the sEl environment.

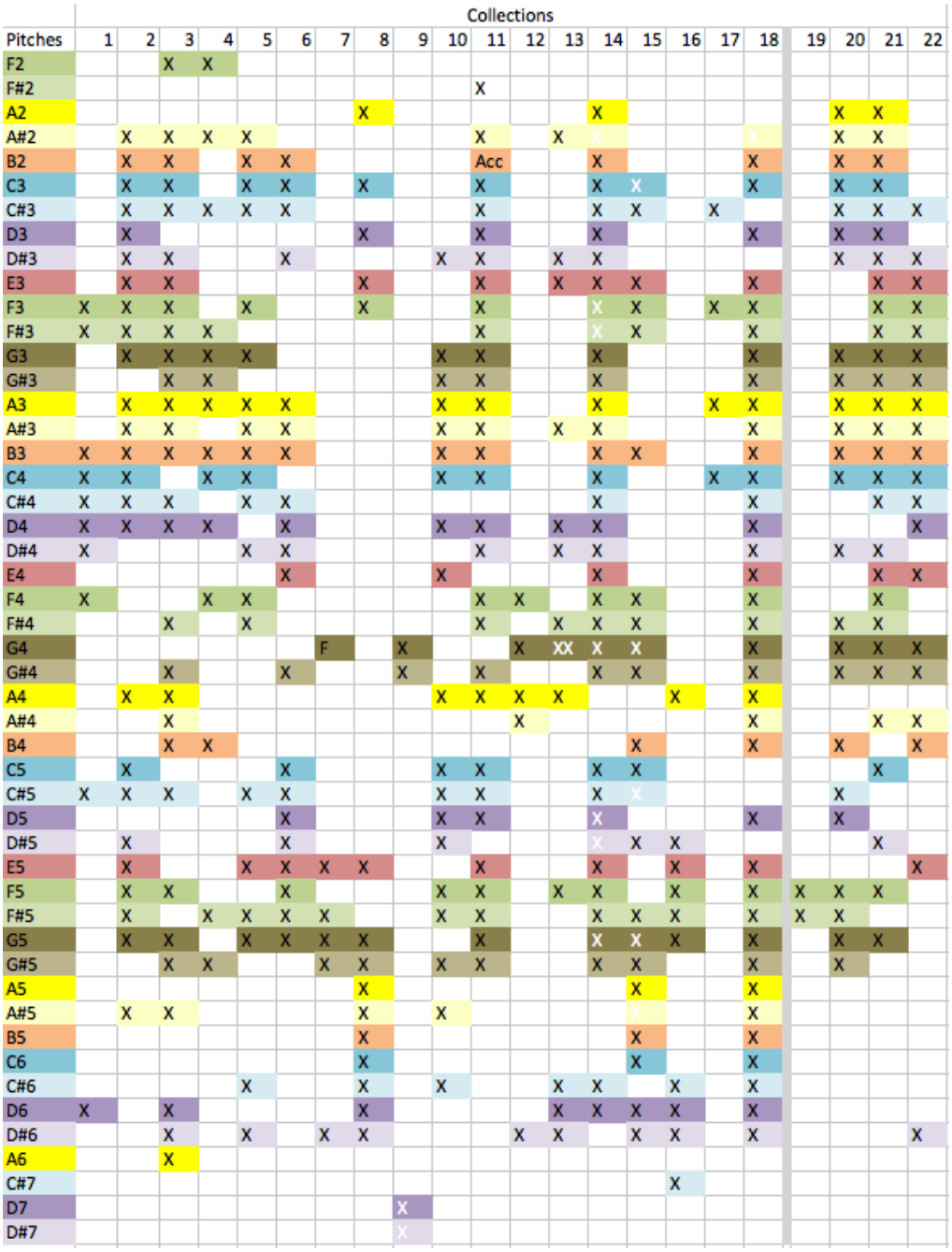


Figure 4.2: Pitch information extracted by sEl from the recordings for the *IgnisFatuus'* fifth movement

Chapter 5

Postlude

In this chapter we will summarize the findings and conclusions from the work on this project, and then sketch a road map for future directions of development.

5.1 Conclusion

The motivation for this project goes back to the days before I started my graduate study at the University of Virginia. Back then I developed a C++ application called *wEave* to assist in the compositional use of my personal library of recordings. It certainly lacked many features of sEl (I should not even compare these two programs), but it had a conceptual structure of informational organization which greatly optimized navigation through the library. It also had a compositional module allowing for some combinatorial strategies which involved sampled materials. I used *wEave* to assist in a few algorithmic compositional projects, the most complex of which was *The Jubjub's Chronicles* (2006), which used about 500 separate sound clips selected from about 15,000 contained in the library. There was no doubt that the utility of such tools will only grow with time due to the rapidly increasing size of personal recording libraries. However, I decided to abandon further development of *wEave* because its design was not systematic and it had reached its ceiling of

scalability.

The years of graduate study allowed the conceptual side of the project to mature and also, as was mentioned in Chapter 1, the open-source music technology tools had evolved tremendously and made it possible to approach this work with greater rigor.

Among the aspects that proved beneficial, one was the decision to base the system's design on the ontological topology of music (1.1). This created a comprehensive perspective for approaching the general structure of the system and deliberation of its functional modules. Recognizing the indispensability of *EncycloSpace* (1.1.5) in the musical process and, most notably, respecting the equi-importance of *receptive* and *productive* modes of navigation helped me find ways to overcome the limiting view of the database as a composing-time-read-only entity. Taking *EncycloSpace*'s dynamicity as semiotic process further aided in the development of recombinance methods of the *compositional subsystem* (2.5.2, 3.4).

The concept of *NOTA-transform* (2.5.2.2) proved to be instrumental as it helped to break down the compositional strategy problematics into smaller analytic parts with mutually independent semantics. Isolating *navigational, sequential* and *temporal* aspects of the EncycloSpace's utilization allowed to devise a clear and scalable structure of the corresponding software modules.

The decision to design sEl as an integrated system played a critical role in allowing this project to attain functional fullness within the designated time. Even though it is still more of a prototype, all major parts of the framework are in place, the system functions as a whole and its components are structurally set for further extensions. This would not be possible without delegating parts of the job to third-party toolsets,

especially in the area of musical informatics. Also, since Python is not the fastest language, the audio functionality vastly benefitted from using the SuperCollider server as a real-time audio engine.

Modular architecture as a design principle is nothing new, but it found itself at home with our project. The system's open design stimulates unlimited extensibility which has proven to be an important element of the compositional software. On a number of occasions while composing music with sEl and driven by specific compositional needs, I found myself adding necessary algorithms to the *NOTA-transform* stages without changing its principal architecture. So, the system can grow on demand according to the user's conceptual perspective.

Despite optimization via the integration with third-party tools, the most time-consuming part of the development process turned out to be constructing the *analysis subsystem* (2.4, 3.3) and putting together the database schema (2.3, 3.2). In the EncycloSpace context, it was the construction of the "mental organization" (1.1.5) of the sound information — the phase of work which stipulated degrees of affordability of that information for the compositional stage.

Since one of the main aspirations for this project was an attempt to upgrade the design of corpus-based compositional software from a *synthesis*-oriented paradigm to a *composition*-oriented one, an assessment of its validity cannot be done without estimating how close we came to the above objective. This, in turn, might be logical to evaluate by looking specifically at the affordances of the compositional part. According to *NOTA-transform*, such an assessment has to consider four separate but coordinated areas: EncycloSpace *navigation*, elements *ordering*, *temporalization* and *adaptation*. Compared to *concatenative synthesis* and *music mosaicing* tools,

we observe the most difference in the second and third stages. Decision-making in sEl's *ordering* and *temporalization* stages is not predicated on any specific patterns of causality. The closest application to sEl conceptually seems to be *AudioGuide* ([56]) but its unavailability did not let us to undertake any detailed analysis.

In the current scope of implementation, sEl is most suitable for acousmatic composition. The possibility of expanding functionality into other compositional areas is addressed below in 5.2.

5.2 Future Work

This project can be expanded in a number of directions. Below we sketch out some of the expansions, improvements and optimizations that we would like to address in future work.

5.2.1 Expansions

5.2.1.1 Sound material types

sEl was initially developed primarily to address the compositional usage of recorded sounds. However, the open architecture allows to easily overcome this quasi-limitation and develop classes to host other genotypes of element production such as sound synthesis.

5.2.1.2 Rendering types

At the moment sEl delivers composition as a waveform. In the future, rendering will be revised in a format-independent way to allow for different output domains: waveform, symbolic notation, control data etc. Also it will be possible to assign spe-

cific output domains to particular compositional objects to allow for hybrid rendering which can be useful for compositions that mix instrumental and acousmatic sound sources.

5.2.1.3 Modality types

This is a far-reaching goal: to add more modalities to the system. In this view, the software becomes a cross-modal compositional framework and EncycloSpace turns into a multi-modal information base. Due to the semiosis of EncycloSpace, elements of different modalities will be able to communicate and thus inform decision-making across the modality border. Modalities can include visual data, textual data, scientific data, network data or pretty much anything else.

5.2.1.4 Compositional vocabulary

The development of sEl is in a prototype stage. Many features of the compositional API have still to be developed. For example, in all phases of *NOTA-trasnform* there are openings to be filled by specific algorithms of navigation, ordering and temporalization. Expansion of the compositional vocabulary is going to be addressed next in the program's future development.

5.2.1.5 Element matching

In most of the corpus-based applications, including sEl, element matching is approached as a *one-to-one* relationship. It seems beneficial though, to expand matching schemas by adding *one-to-many* scenarios. This can substantially increase the diversity of matching results. In such *aggregate elements matching* scenario several elements cooperatively represent another element. This cooperation can be vertical, i.e. across spectrum, and/or temporal. This expansion is inspired by the proposition described in [56].

5.2.1.6 Elements containment

Currently there is one set of child elements per each parent element. This can be expanded to allow for different subdivisions of the same element. Such a change should improve elements representations on different time scales.

5.2.1.7 Data visualization

The *matplotlib* library, currently used for visualization, is not very well equipped for displaying dynamically updated data. It saved some development time in the process, but in the future it should be replaced by hardware-accelerated visualization. OpenGL based libraries like *Galry* [98] or *VISPY* [29] seem to be good candidates.

Besides replacing the visualization library, new visualization modules should be written to visualize:

- All element's descriptors at once
- All characteristic values of the element's dynamic descriptor in a single visualization block
- Analysis graph
- Composition content

5.2.1.8 User interface and interactivity

At the moment, the usage of the *compositional subsystem* is done via scripting in the code base and then running the software to compute the sound results. It would be beneficial to add interactive elements to the user interface to allow:

- Display of the library of compositional objects
- Construction of the composition at run-time by scripting or visual programming
- Interactive assembling of analysis graphs by methods of visual programming

- Interactive testing and reprogramming of compositional substructures
- *more...*

5.2.1.9 Documentation and availability

To make this project useful for a wide audience, some time will be needed to create user documentation and examples. After critical improvements and optimizations the software's code base will be uploaded to one of the publicly accessible repositories.

5.2.2 Optimizations

There are areas where the current implementation can be optimized, namely

- Audio data allocations. These are currently administered by each sound element independently but should be managed from central location.
- Speed of database reflection. There can be hundreds of thousands of elements and millions of extracted descriptors in the database. The database browser needs to be optimized in order to decrease delays when database information is being harvested for display.
- Speed of descriptor computation. Optimization of analysis computation can be done by splitting the task into smaller chunks which can be separately computed on different process threads or, for large number of elements, be scheduled for cluster computing if it is available.

When sEl's implementation reaches a degree of complete functional realization, it will be ported to a faster language platform such as C++.

A summarized conclusion can be expressed as a belief, drawn from the experience of working on this project, that it can furnish additional attention to the problematics of the design of sample-based compositional methodologies and, hopefully, it will also

be able to contribute a number of constructive solutions to both theoretical and practical aspects of the topic.

Appendix A

Default descriptors

Table A.1: Temporal descriptors

Feature	Data type	Dynamic	Range
Timescale	symbol		[0,12]
Event cardinality	integer		[1,∞]
Temporal density	real		[0,1]
Onset cardinality	integer		[1,∞]
Dynamic complexity	real		[0,100]
Inter-onset intervals	real[]	X	[0,dur]
Q-time	real		[0,dur)

Table A.2: Energy descriptors

Feature	Data type	Dynamic	Range
Dynamics	symbol		[0,10]
ELF loudness	real[]	X	[-100,0]
Start bark bands energy	real[27]		[0,1]
Q-time bark bands energy	real[27]		[0,1]
End bark bands energy	real[27]		[0,1]
Start spectral RMS	real		[0,1]
Q-time spectral RMS	real		[0,1]
End spectral RMS	real		[0,1]

Table A.3: Tonal descriptors

Feature	Data type	Dynamic	Range
Pitchiness	real[3]		[0,1]
Pitches cardinality	integer		[1,∞]
Pitches	real[]		[0,127]
Pitch durations	real[]		[0,dur]
Pitch quotas	real[]		[0,1]
Starting pitch	real		[0,127]
Ending pitch	real		[0,127]
Momentary pitches cardinality	integer		[1,∞]
Momentary pitches	real[]		[0,127]
Momentary pitches duration	real		[0,dur]
Momentary pitches quota	real		[0,1]
Chroma cardinality	integer		[1,∞]
Chromas	symbol[]		[0,11]
Chroma durations	real[]		[0,dur]
Chroma quotas	real[]		[0,1]
Global chroma simultaneity ratio	real		[0,1]
Local chroma simultaneity ratio	real		[0,1]
Maximum chroma simultaneity	integer		[1,∞]
Maximum chroma simultaneity local ratio	real		[0,1]

Table A.4: Rhythm descriptors

Feature	Data type	Dynamic	Range
CPMs	real[]		[0,640]
CPM strengths	real[]		[0,1]
CPM evolve	real	X	[0,640]
Tatum	real		[0,2000]

Table A.5: Spectral descriptors

Feature	Data type	Dynamic	Range
Spectral centroid	real	X	[0,Nyquist]
Spectral complexity	real	X	[0,100]
Spectral decrease	real	X	[-∞,∞]
Spectral flux	real	X	[0,1]
Spectral peakiness	real	X	[-∞,∞]
Spectral rolloff	real	X	[0,Nyquist]
Spectral skewness	real	X	[-∞,∞]
Spectral spread	real	X	[0,Nyquist/2]

Appendix B

Default category taxonomy

This appendix contains description of the default taxonomy of sEl's categories.

There are six top categories:

1. **Morphology** — describes sound sources as entities
2. **Source** — describes material aspects of vibrating bodies
3. **Resonator** — describes material aspects of resonating bodies
4. **Instrument-wise** — describes aspects of sound source as instrument
5. **Play-wise** — describes performing aspects of sound production
6. **Spectromorphology** — describes phenomenological aspects of sound

B.1 Morphology categories

- Acoustic
  - Human
    - Body
    - Voice
  - Manmade
    - Instrument

- Aerophone
  - Accordion
  - Brass
    - Cimbasso
    - Double horn
    - Euphonium
    - Flugelhorn
    - etc
  - Organ
    - Electric organ
    - Pipe organ
    - Reed organ
    - etc
  - Woodwinds
    - Bassoon
    - Clarinet
    - Contrabassoon
    - English horn
    - Flute
    - etc
- Aquaphone
- Chordophone
  - Banjo
  - Dulcimer
  - Guitar
  - etc

    - Keyboard
        - Clavinet
        - Harpsichord
        - Piano
    - Strings
        - Cello
        - Double bass
        - Viola
        - Violin
- Idiophone
    - Pitched
        - Agogo
        - Almglocken
        - Bian qing
        - Bianzhong
        - etc
    - Unpitched
        - Anvil
        - Bell tree
        - Bells
        - Cabasa
        - etc
- Membranophone
    - Berkete
    - Bodhran
    - Bongo

                - Boobam
                - etc
            - Pyrophone
        - Object
    - Nature
        - Phenomena
            - Fire
            - Rain
            - Water
            - Wind
            - etc
        - Lifeforms
            - Plants
            - Animals
- Synthetic
    - Instrument
        - Analog
        - Digital
        - Virtual
    - Soundobject

B.2 Source categories

- Form factor
    - Bell
    - Bellows

    - Block
    - Cone
    - Conical bore
    - Cube
    - Cylinder
    - Cylindrical bore
    - Disc
    - Hollow object
    - Membrane
    - Particles
    - Pipe
    - Reed
    - Rod
    - Sheet
    - Sphere
    - Stick
    - String
    - Stripe
    - etc
- State of matter
    - Fire
    - Gas
    - Liquid
    - Plasma
    - Solid

B.3 Resonator

- Ceramic
- Glass
- Metal
- Plastic
- Rubber
- Skin
- Stone
- Wood

B.4 Instrument-wise

- Beater
    - Brush
    - Hammer
    - Hand
        - Fingers
        - Palm
    - Mallet
        - Plastic
        - Yarn
        - etc
- Group
    - Ethnic
    - General
    - Historical

        - Nontraditional
        - Orchestral
    - Location
        - Center
        - Edge
        - Middle
        - Rim
        - Side
    - Modification
        - Mute
            - Bucket
            - Cup
            - Hand
            - etc
        - Sizzle
        - Sordino
        - Stopped
        - etc
    - Range
        - Alto
        - Bariton
        - Bass
        - Contrabass
        - Piccolo
        - Soprano

    - Tenor
- State
    - Closed
    - Open
    - etc

B.5 Play-wise

- Articulation
    - Arpeggio
    - Attack
    - Bend
    - Bisbigliando
    - Detachet
    - Expressive
    - Flatter
    - Fluttertongue
    - Harmonics
        - Aritifical
        - Natural
    - Jazz tongued
    - Legato
    - Marcato
    - Modulation
        - Shake
        - Tremolo

        - Trill
        - Vibrato
    - Pizzicato
        - Bartok
        - Snap
    - Portando
    - Sforzando
    - Staccatissimo
    - Staccato
- Excitation
    - Blown
    - Bowed
    - Hit
    - Plucked
    - Rubbed
    - Scraped
    - etc
- Multiplicity
    - Duo
    - Ensemble
    - Octet
    - Orchestra
    - Quartet
    - Quintet
    - Section

        - Septet
        - Sextet
        - Solo
        - Trio
    - Sequence
    - Speed
        - Change
            - Accelerando
            - Rallentando
        - Level
            - Fast
            - Slow
    - Technique
        - Brush
        - Choke
        - Click
        - Col legno
        - Hit
        - Overblown
        - Pluck
        - Pres de la table
        - Ride
        - Roll
        - Saltando
        - Scrape

        - Shot
        - Slap
        - Slash
        - Strike
        - Sul ponticello
        - Sul tasto
        - Tap
        - etc
    - Tempo
        - Adagietto
        - Adagio
        - Allegretto
        - Allegro
        - etc
        - Vivace
        - Vivacissimo

B.6 Spectromorphology

- Character
    - Breathy
    - Crisp
    - Dirty
    - Drone
    - Growl
    - etc

- Dynamics
    - Change
        - Crescendo
        - Diminuendo
    - Level
        - f
        - ff
        - etc
        - ppppp
- Motion
    - Direction
        - Dextrally
        - Down
        - Inward
        - Outward
        - Sinistrally
        - Up
    - Speed
    - Type
        - Fall
        - Glide
        - Glissando
        - Rip
        - Slide
        - Smear
        - etc

- Texture
    - Dense
    - Dynamic
    - Granular
    - Monotonous
    - Sparse
    - etc

Appendix C

Comparison of External MIR Toolsets

In tables toolsets are encoded as

- A — aubio [24]
- B — Bregman toolbox [32]
- C — Echonest [44]
- E — Essentia [22]
- L — LibXtract [26]
- M — Marsyas [117]
- Y — YAAFE [75]

Table C.1: Toolsets Input/Output

Functionality	A	B	C	E	L	M	Y
Audio input	x	x	x	x	x	x	x
Audio output	x	x	x	x	x	x	x
Features input		x	x	x (YAML)	x	x	x (HDF5)
Features output	x	x	x	x (YAML)	x	x	x (HDF5)
Database support		x					

Table C.2: Features (Amplitude)

Functionality	A	B	C	E	L	M	Y
Envelope				x			x
Envelope centroid							x
Envelope flatness				x			
Envelope kurtosis							x
Envelope skewness							x
Envelope spread							x
Log Attack time				x			
Max amp to total length				x			
Min amp to total length				x			

Table C.3: Features (Chroma, harmony, key)

Functionality	A	B	C	E	L	M	Y
Chord detection				x			
Chords change rate				x			
Chords histogram				x			
Chords integration				x			
Chords Key				x			
Chords progression				x			
Chroma		x				x	
Dissonance Measure		x		x			
Equal-tempered deviation				x			
Harmonic Peaks				x			
Harmonic Pitch Class Profile (HPCP)				x			
Inharmonicity				x	x		
Key detection			x	x			
Log OBSI Ratio							x
Non-tempered energy ratio				x			
Non-tempered peaks energy ratio				x			
Octave Band Signal Intensity (OBSI)							x
Odd-to-Even Harmonic Energy ratio				x	x		
Scale detection			x	x			
Tonic detection				x			
Tuning frequency				x			
Tonality Coefficient					x		
Tristimulus				x	x		

Table C.4: Features (Energy)

Functionality	A	B	C	E	L	M	Y
Band energy				x			
Band Energy ratio				x			
Bands array energy				x			
dbPower		x			x		
Replay Gain				x			
RMS energy		x		x	x		
Total energy			x	x			x

Table C.5: Features (Frequency, pitch)

Functionality	A	B	C	E	L	M	Y
Fundamental frequency (F0)	x		x	x	x	x	
Max magnitude Frequency				x			
Melody contour		x		x			
Multipitch	x						
Noisiness					x		
Pitch contours				x			
Pitch salience				x			
PitchContourRatio				x			
Pitchiness				x			
Zero-crossing rate				x	x	x	x

Table C.6: Features (Perceptual energy)

Functionality	A	B	C	E	L	M	Y	
Bark bands					x	x		x
Equivalent sound level (Leq)					x			
ERB bands					x			
Long-term Loudness (LARM)					x			
Mel bands					x			
Perceptual Sharpness						x		x
Perceptual Spread								x
Total loudness				x	x	x		

Table C.7: Features (Spectral)

Functionality	A	B	C	E	L	M	Y
	Spectral periodicity						
Auto-correlation				x	x		x
Gammatone feature cepstrum coefficients (GFCC)				x			
Linear Prediction Coefficients				x		x	x
Mel-frequency cepstral coefficients MFCCs		x		x	x	x	x
	Spectral shape						
High Frequency content (HFC)				x			
Spectal Contrast				x			
Spectral Centroid		x		x	x	x	x
Spectral Complexity				x			
Spectral Crest					x	x	
Spectral Decrease							x
Spectral Flatness					x	x	x
Spectral Kurtosis				x	x		x
Spectral Peaks				x	x		
Spectral Roll-off				x	x	x	x
Spectral Skewness				x	x		x
Spectral Slope					x		x
Spectral Spread		x		x	x		x
	Spectral temporal						
Spectral Flux				x		x	x
Spectral Variation							x

Table C.8: Features (Temporal)

Functionality	A	B	C	E	L	M	Y
Beat loudness				x			
Beat tracking (Tempo detection)	x		x	x		x	
BPM harmonics				x			
BPM histogram				x			
Danceability			x	x			
Dynamic Complexity				x			
Effecftive duration				x			
Grain detection							x
Irregularity					x		
Large-scale Structure detection			x				
Liveness detection			x				
Measures detection			x				
Novelty curve				x			
Onset detection	x			x			x
Onsets rate				x			
Rubato detection				x			
Segmentation	x	x	x	x			
Smoothness					x		
Tatum detection			x				
Temporal Centroid				x			
Time signature detection			x	x			
Transients separation	x						

Table C.9: Features (Volume)

Functionality	A	B	C	E	L	M	Y	
Fade in detection				x	x			
Fade out detection				x	x			
Silence detection	x				x			
Silence rate					x			
Strong Decay					x			
Tremolo detection								x

Table C.10: Features (Other)

Functionality	A	B	C	E	L	M	Y
Quefrency (Cepstrum) Chroma		x					
Source detection			x				
Speech detection			x				

Table C.11: Transforms

Functionality	A	B	C	E	L	M	Y
Cepstrum							x
Constant-Q transform		x					
Discrete Cosine Transform				x	x		
Harmonic Spectrum					x		
Inverse FFT				x			
Magnitude spectrum				x			x
Mel spectrum							x
Power spectrum		x		x			
STFT (complex spectrum)		x		x	x		
Rhythm transform				x			

Table C.12: Auxiliary processing

Functionality	A	B	C	E	L	M	Y
All-pass filter				x			
Band-pass filter				x			
Band-reject filter				x			
DC removal				x			
Framing				x			x
Equal Loudness Filtering				x			
High-pass filter				x			
IIR filter				x			
Low-pass filter				x			
Metadata reader				x			
Moving average filter				x			
Multiplexing				x			
Noise adder				x			
Phase vocoder	x						x
Resampling				x			
Spectral Whitening				x			
Stereo Demuxing				x			
Test signals generation		x					
Tuning systems		x					
Visualization		x				x	
Windowing				x			

Table C.13: Additional algorihms

Functionality	A	B	C	E	L	M	Y
Central Moments				x	x		
Crest computation				x			
Average Magnitude Difference Function (AMDF)					x		
Average Squared Difference Function (ASDF)					x		
Bhattacharyya distance		x					
Cosine distance		x					
Cross correlation				x			
Decrease computation				x			
Derivatives				x			x
Dynamic Time Warping		x					
Flatness				x			
Histogram							x
KMeans clustering		x					
Kullback-Leibler divergence		x		x			
Multidimensional scaling		x					
Multivariate Gaussian model		x					
Normed dot-product distance		x					
Peak detection				x			
Pearson product-moment correlation coefficient		x					
Primary Component Analysis (PCA)				x			
Probabilistic Latent Component Analysis (PLCA)		x					
Shift-Invariant PLCA (SI-PLCA)		x					
Slope							x
Support Vector Machines (SVM)						x	

Appendix D

Dependencies

This project integrates or depends on the functionality of the following open-source frameworks and libraries:

aubio (aubio.org) — audio analysis library

Essentia (essentia.upf.edu) — audio analysis library

matplotlib (matplotlib.org) — Python plotting library

NumPy (numpy.org) — N-dimensional array processing

PostgreSQL (postgresql.org) — SQL database management system

PySide (pyside.org) — Python bindings for the Qt framework

Qt (qt.io) — multi-platform software development framework

SC (pypi.python.org/pypi/SC/0.2) — Python interface to SuperCollider server

SciPy (scipy.org) — Python scientific computing

SQLAlchemy (sqlalchemy.org) — Python SQL toolkit and Object Relational Mapper

SuperCollider (supercollider.github.io) — audio synthesis and algorithmic composition language

Bibliography

[1] *ISMIR 2009: Proceedings of the 10th International Society for Music Information Retrieval Conference*. ISMIR, Kobe, Japan.

[2] *ISMIR 2010: Proceedings of the 11th International Society for Music Information Retrieval Conference*. ISMIR, Utrecht, Netherlands.

[3] *ISMIR 2011: Proceedings of the 12th International Society for Music Information Retrieval Conference*. University of Miami, Florida.

[4] *ISMIR 2012: Proceedings of the 13th International Society for Music Information Retrieval Conference*. Mosteiro S.Bento Da Vitória.

[5] *ICMC International Computer Music Conference*. Hong Kong University of Science and Technology, China, 1996.

[6] *ISMIR 2002: Proceedings of the 3rd International Society for Music Information Retrieval Conference*. Paris, France, October 2002.

[7] *ISMIR 2007: Proceedings of the 8th International Society for Music Information Retrieval Conference*. Vienna, Austria, September 2007.

[8] *The Music Ontology, musicontology.com/specification/*, 2015. Accessed: 2015-05-08.

[9] ISO/IEC 15938-4. Information technology – multimedia content description interface – part 4: Audio, 06-15 2002.

[10] Carlos Agon, Gerard Assayag, and Jean Bresson. *OpenMusic, repmus.ircam.fr/openmusic/home*, 2015. Accessed: 2015-05-13.

[11] Xavier Amatriain, Jordi Bonada, Alex Loscos, Josep Luis Arcos, and Vincent Verfaille. Content-based transformations. *Journal of New Music Research*, 32:(95–114), 2003.

[12] Christopher Ariza. Navigating the landscape of computer aided algorithmic systems: A definition, seven descriptors, and a lexicon of systems and research, 2005.

[13] Christopher Ariza. *An Open Design for Computer-Aided Algorithmic Music Composition: athenaCL*. PhD thesis, New York University, September 2005.

[14] Maria Camila Barioni, Humberto Razente, Caetano Traina Jr., and Agma Traina. Querying complex objects by similarity in SQL.

[15] Luke Barrington, Antoni B. Chan, and Gert Lanckriet. Modeling music as a dynamic texture. *IEEE Transactions On Audio, Speech, And Language Processing*, 18(3):602–612, 2010.

[16] Roland Barthes. *Elements of semiology*. Macmillan, 1977.

[17] Mike Bauer. *SQL Alchemy — The Database Toolkit for Python, www.sqlalchemy.org*, 2015. Accessed: 2015-05-03.

[18] Bernard Bel. Migrating musical concepts: An overview of the bol processor. *Computer Music Journal*, 2(2):56–64, 1998.

[19] Thierry Bertin-Mahieux, Daniel P. W. Ellis, Brian Whitman, and Paul Lamere. The million song dataset. In *ISMIR 2011: Proceedings of the 12th International Society for Music Information Retrieval Conference* [3].

[20] Jeffrey Adam Bilmes. Timing is of the essence: Perceptual and computational techniques for representing, learning, and reproducing expressive timing in percussive rhythm. Master's thesis, Massachusetts Institute of Technology, 1993.

[21] Sturm Bob. Matconcat: An application for exploring concatenative sound synthesis using matlab. In *ICMC International Computer Music Conference*. ICMC, Miami, Florida, 2004.

[22] Dmitry Bogdanov, Nicolas Wack, Emilia Gomez, Sankalp Gulati, Perfecto Herrera, Oscar Mayor, Gerard Roma, Justin J. Salamon, Jose Zapata, and Xavier Serra. ESSENTIA - an audio analysis library for music information retrieval. pages 493–498, 2013.

[23] Albert S. Bregman. *Auditory Scene Analysis. The perceptual organization of sound*. A Bradford Book, The MIT Press, Cambridge, Massachusetts, 1990.

[24] Paul Brossier. *aubio: a tool for annotating audio signals, aubio.org*, 2015. Accessed: 2015-05-26.

[25] Paul M. Brossier. *Automatic Annotation of Musical Audio for Interactive Applications*. PhD thesis, Queen Mary, University of London, Queen Mary, University of London, 2006.

[26] Jamie Bullock. Libxtract: A lightweight library for audio feature extraction. In *ICMC International Computer Music Conference*. ICMC, Copenhagen, Denmark, 2007.

[27] Phil Burk, Larry Polansky, and David Rosenboom. *HMSL: the Hierarchical Music Specification language, www.softsynth.com/hmsl/*, 2015. Accessed: 2015-05-13.

[28] Andres Cabrera. *CsoundQt, qutecsound.sourceforge.net*, 2015. Accessed: 2015-05-13.

[29] Luke Campagnola, Almar Klein, Cyrille Rossant, and Nicolas Rougier. Vispy, a modern and interactive scientific visualisation.

[30] Scaletti Carla and Kurt Hebel. *Kyma: sound design inspiration, kyma.symbolicsound.com*, 2015. Accessed: 2015-05-13.

[31] Michael Casey. Audiodb: Scalable approximate nearest-neighbor search with automatic radius-bounded indexing. *Acoustical Society of America*, 124(4):2571, 2008.

[32] Michael Casey. *Bregman Audio-Visual Information Toolbox: Advanced Tools for the Digital Arts and Humanities, digitalmusics.dartmouth.edu/bregman/*, 2015. Accessed: 2015-06-03.

[33] Carmine Emanuele Cella. Sound-types: A new framework for symbolic sound analysis and synthesis. In *ICMC International Computer Music Conference*. ICMC, Huddersfield, UK, July 2011.

[34] Carmine Emanuele Cella and J. J. Burred. Advanced sound hybridizations by means of the theory of sound-types. 2013.

[35] The Qt Company. *Qt: cross-platform application and UI development framework, www.qt.io/qt-framework/*, 2015. Accessed: 2015-05-03.

[36] David Cope. *Experiments in Musical Intelligence*. A-R Editions, 1996.

[37] Roger B. Dannenberg. The implementation of nyquist, a sound synthesis language. *Computer Music Journal*, 21(3):71–82, 1997.

[38] Ferdinand De Saussure and Wade Baskin. *Course in general linguistics*. Columbia University Press, 2011.

[39] Nick Didkovsky. *JMSL: Java Music Specification Language, www.algomusic.com/jmsl/*, 2015. Accessed: 2015-05-13.

[40] Simon Dixon. Onset detection revisited. In *DAFx International Conference on Digital Audio Effects*, pages 133–137. DAFx.

[41] Simon Dixon, Fabien Gouyon, and Gerhard Widmer. Towards characterisation of music via rhythmic patterns. In *ISMIR 2004: Proceedings of the 5th International Society for Music Information Retrieval Conference*. Barcelona, Spain, October 2004.

[42] J Stephen Downie. Music information retrieval. *Annual review of information science and technology*, 37(1):295–340, 2003.

[43] Jos J. Eggermont. Between sound and perception - reviewing the search for a neural code. *Hearing Research*, 157:1–42, 2001.

[44] Daniel PW Ellis, Brian Whitman, Tristan Jehan, and Paul Lamere. The echo nest musical fingerprint. In *ISMIR 2010: Proceedings of the 11th International Society for Music Information Retrieval Conference* [2].

[45] Archer Endrich. Composers' desktop project: a musical imperative. *Organised Sound*, 2(1):29–33, 1997.

[46] Florian Eyben, Sebastian Böck, Björn Schuller, and Alex Graves. Universal onset detection with bidirectional long short-term memory neural networks. In *ISMIR 2010: Proceedings of the 11th International Society for Music Information Retrieval Conference* [2].

[47] Jonathan Foote, Matthew L. Cooper, and Unjung Nam. Audio retrieval by rhythmic similarity. In *ISMIR 2002: Proceedings of the 3rd International Society for Music Information Retrieval Conference* [6].

[48] Takuya Fujishima. Realtime chord recognition of musical sound: A system using common lisp music. In *Proc. ICMC*, volume 1999, pages 464–467. Bejing, China, 1999.

[49] Richard Gael, Mathieu Ramona, and Slim Essid. Combined supervised and unsupervised approaches for automatic segmentation of radiophonic audio streams. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, volume II, pages 461–464. IEEE.

[50] Jorg Garbers. An integrated MIR programming and testing environment. In *ISMIR 2006: Proceedings of the 7th International Society for Music Information Retrieval Conference*. University of Victoria, Canada, October 2006.

[51] Brad Garton, John Gibson, Doug Scott, and Dave Topper. *RTcmix: an open-source digital signal processing and sound synthesis language, http://rtcmix.org*, 2015. Accessed: 2015-05-13.

[52] Emilia Gómez. Tonal description of polyphonic audio for music content processing. *INFORMS Journal on Computing*, 18(3):294–304, 2006.

[53] The HDF group. *The HDF Group — Information, Support and Software, www.hdfgroup.org*, 2015. Accessed: 2015-05-03.

[54] The PostgreSQL Global Development Group. *PostgreSQL: The world's most advanced open-source database, www.postgresql.org*, 2015. Accessed: 2015-05-04.

[55] Enric Guaus and Perfecto Herrera. The rhythm transform: Towards a generic rhythm description, 2004.

[56] Benjamin Hackbarth, Norbert Schnell, and Diemo Schwarz. Audioguide: A framework for creative exploration of concatenative sound synthesis. *IRCAM research report*, 2011.

[57] Philippe Hamel, Yoshua Bengio, and Douglas Eck. Building musically-relevant audio features through multiple timescale representations. In *ISMIR 2012: Proceedings of the 13th International Society for Music Information Retrieval Conference* [4].

[58] Steven Hazel. *Soundmosaic, awesame.org/soundmosaic/*, 2015. Accessed: 2015-05-13.

[59] Lejaren Hiller and Leonard Isaacson. Musical composition with a high-speed digital computer. *Journal of the Audio Engineering Society*, 6(3):154–160, 1958.

[60] Jason Hockman, Matthew E. P. Davies, and Ichiro Fujinaga. One in the jungle: Downbeat detection in hardcore, jungle, and drum and bass. In *ISMIR 2012: Proceedings of the 13th International Society for Music Information Retrieval Conference* [4].

[61] Peter Hoffmann. *Music Out of Nothing? A Rigorous Approach to Algorithmic Composition by Iannis Xenakis*. PhD thesis, der Technischen Universitaet Berlin, April 2009.

[62] Eric J. Humphrey, Juan Pablo Bello, and Yann LeCun. Moving beyond feature design: Deep architectures and automatic feature learning in music informatics. In *ISMIR 2012: Proceedings of the 13th International Society for Music Information Retrieval Conference* [4].

[63] Tristan Jehan. *Creating Music by Listening*. PhD thesis, Massachusets Institute of Technology, 2005.

[64] Kristoffer Jensen and Tue Haste Andersen. Beat estimation on the beat. In *Applications of Signal Processing to Audio and Acoustics, 2003 IEEE Workshop on.*, pages 87–90. IEEE, 2003.

[65] Gottfried Michael Koenig. Project 2. *Electronic Music Report*, 3, 1970.

[66] Gottfried Michael Koenig. Project one. *Electronic Music Report*, 2, 1970.

[67] Olivier Lartillot and Petri Toiviainen. MIR in matlab (II): A toolbox for musical feature extraction from audio. In *ISMIR 2007: Proceedings of the 8th International Society for Music Information Retrieval Conference* [7].

[68] Cyril Laurier, Owen Meyers, Joan Serrà, Martin Blech, Perfecto Herrera, and Xavier Serra. Indexing music by mood: Design and integration of an automatic content-based annotator. *Multimedia Tools and Applications*, 48(1):161–184, 2010.

[69] Mikael Laurson and Mika Kuuskankare. *PWGL: A Visual Programming Language for Music and Sound, http://www2.siba.fi/PWGL/*, 2015. Accessed: 2015-05-13.

[70] Arie A. Livshin, Geoffroy Peeters, and Xavier Rodet. Studies and improvements in automatic classification of musical sound samples. In *ICMC International Computer Music Conference*. ICMC, Singapore, Singapore, 2003.

[71] Matija Marolt. Probabilistic segmentation and labeling of ethnomusicological field recordings. In *ISMIR 2009: Proceedings of the 10th International Society for Music Information Retrieval Conference* [1].

[72] Paul Masri and Andrew Bateman. Improved modelling of attack transients in music analysis-resynthesis. In *ICMC International Computer Music Conference* [5], pages 100–103.

[73] Max Mathews. What is loudness? In Perry Cook, editor, *Music, Cognition, and Computerized Sound. An Introduction to Psychoacoustics*, chapter 6, pages 71–78. The MIT Press, Cambridge MA, 2001.

[74] Benoît Mathieu, Slim Essid, Thomas Fillon, Jacques Prado, and Gaël Richard. Yaafe, an easy to use and efficient audio feature extraction software. In *ISMIR 2010: Proceedings of the 11th International Society for Music Information Retrieval Conference* [2].

[75] Benoit Mathieu, Slim Essid, Thomas Fillon, Jacques Prado, and Gael Richard. YAAFE, an easy to use and efficient audio feature extraction software. 2010.

[76] Guerino Mazzola. *The Topos of Music: Geometric Logic of Concepts, Theory, and Performance*. Birkhäuser, 2002.

[77] James McCartney. Supercollider: a new real time synthesis language. In *ICMC International Computer Music Conference* [5].

[78] Jon McCormack. *Nodal: Generative Music Software, www.csse.monash.edu.au/ cema/nodal/*, 2015. Accessed: 2015-05-13.

[79] Daniel McEnnis, Cory McKay, Ichiro Fujinaga, and Philippe Depalle. jaudio: A feature extraction library. In *ISMIR 2005: Proceedings of the 6th International Society for Music Information Retrieval Conference*. London, UK, September 2005.

[80] Cory McKay and Ichiro Fujinaga. jwebminer: A web-based feature extractor. In *ISMIR 2007: Proceedings of the 8th International Society for Music Information Retrieval Conference* [7].

[81] Adrien Merer, Sølvi Ystad, Richard Kronland-Martinet, and Mitsuko Aramaki. Abstract sounds and their applications in audio and perception research.

In Sølvi Ystad, Mitsuko Aramaki, Richard Kronland-Martinet, and Kristoffer Jensen, editors, *Exploring Music Contents - 7th International Symposium, CMMR 2010, Málaga, Spain, June 21-24, 2010. Revised Papers*, volume 6684 of *Lecture Notes in Computer Science*, pages 176–187. Springer, 2010.

[82] Eduardo Miranda. *Composing Music with Computers*. Focal Press, 2001.

[83] Jean Molino, JA Underwood, and Craig Ayrey. Musical fact and the semiology of music. *Music Analysis*, pages 105–156, 1990.

[84] Jean-Jacques Nattiez. *Music and discourse: Toward a semiology of music*. Princeton University Press, 1990.

[85] Travis E Oliphant. Python for scientific computing. *Computing in Science & Engineering*, 9(3):10–20, 2007.

[86] Tae Hong Park. *Towards automatic musical instrument timbre recognition*. Princeton University, 2004.

[87] Tae Hong Park, Zhiye Li, and Wen Wu. Easy does it: The electro-acoustic music analysis toolbox. In *ISMIR 2009: Proceedings of the 10th International Society for Music Information Retrieval Conference* [1].

[88] Jouni Paulus and Anssi Klapuri. Measuring the similarity of rhythmic patterns. In *ISMIR 2002: Proceedings of the 3rd International Society for Music Information Retrieval Conference* [6].

[89] Geoffroy Peeters. A large set of audio features for sound description (similarity and classification) in the GUIDADO project. April 2004.

[90] Geoffroy Peeters, Bruno L Giordano, Patrick Susini, Nicolas Misdariis, and Stephen McAdams. The timbre toolbox: Extracting audio descriptors from musical signals. *The Journal of the Acoustical Society of America*, 130(5):2902–2916, 2011.

[91] Geoffroy Peeters and Xavier Rodet. Automatically selecting signal descriptors for sound classification. In *ICMC International Computer Music Conference*. ICMC, Goteborg, Sweden, 2002.

[92] Andre Pires and Marcelo Queiroz. Real-time unsupervised music structural segmentation using dynamic descriptors. In *SMC 2011: Sound and Music Computing Conference*. University of Padova, Italy.

[93] Stephen Travis Pope. A taxonomy of computer music. *Contemporary Music Review*, 13(2):137–145, 1996.

[94] Miller Puckette. *MAX/MSP: Visual programming language, cycling74.com/products/max/*, 2015. Accessed: 2015-05-13.

[95] Miller Puckette. *Pure Data: Open-source visual programming language, https://puredata.info/*, 2015. Accessed: 2015-05-13.

[96] Yves Raimond, Samer Abdallah, Mark Sandler, and Frederick Giasson. The music ontology. In *ISMIR 2007: Proceedings of the 8th International Society for Music Information Retrieval Conference* [7].

[97] Curtis Roads. *Microsound*. MIT Press, Cambridge, 2002.

[98] Cyrill Rossant and Kenneth D. Harris. Hardware-accelerated interactive data visualization for neuroscience in python. *Frontiers in Neuroinformatics*, 7(36), 2013.

[99] Bill Schottstaedt. *CLM: Common Lisp Music, ccrma.stanford.edu/software/clm/*, 2015. Accessed: 2015-05-13.

[100] Diemo Schwarz. *Data-Driven Concatenative Sound Synthesis*. PhD thesis, Ircam - Centre Pompidu, Ircam - Centre Pompidu, January 2004.

[101] Diemo Schwarz. Concatenative sound synthesis: The early years. *Journal of New Music Research*, 35:(3–22), 2007.

[102] Diemo Schwarz. *CataRT: Real-Time Corpus-Based Concatenative Synthesis, imtr.ircam.fr/imtr/CataRT*, 2015. Accessed: 2015-05-13.

[103] Diemo Schwarz and Benjamin Hackbarth. Navigating variation: Composing for audio mosaicing. In *ICMC International Computer Music Conference*. ICMC, Ljubljana, Slovenia, September 2012.

[104] Diemo Schwarz and Norbert Schnell. A modular sound descriptor analysis framework for relaxed-real-time applications. In *ICMC International Computer Music Conference*. ICMC, New York City, New York, June 2010.

[105] Diemo Schwarz, Norbert Schnell, and Sebastien Guillini. Scalability in content-based navigation of sound databases. 2009.

[106] Malcolm Slaney and Michael Casey. Locality-sensitive hashing for finding nearest neighbors. *IEEE Signal Processing Magazine*, 2008.

[107] Denis Smalley. The listening imagination: Listening in the electroacoustic era. *Contemporary Music Review*, 13(2):77–107, 1996.

[108] Andrew Sorensen. *Extempore, extempore.moso.com.au*, 2015. Accessed: 2015-05-13.

[109] Yury Spitsyn. *Music States and Decision-making Systems: A Symbolic Framework for Assessment and Comparison of Automated Composition Paradigms*, 2010.

[110] Thomas Stoll. Beyond concatenation: Some ideas for the creative use of corpus-based sonic material. In *ICMC International Computer Music Conference*. ICMC, Montreal, Canada, 2009.

[111] Sebastian Streich. *Music Complexity: A Multi-Faceted Description of Audio Content*. PhD thesis, 2006.

[112] Bob L. Sturm. Adaptive concatenative sound synthesis and its application to micromontage composition. *Computer Music Journal*, 30(4):46–66, 2006.

[113] Chee-Chuan Toh, Bingjun Zhang, and Ye Wang. Multiple-feature fusion based onset detection for solo singing voice. In *ISMIR 2008: Proceedings of the 9th International Society for Music Information Retrieval Conference*. ISMIR, Philadelphia, PA, USA, September 2008.

[114] Barry Truax. The podx system: Interactive compositional software for the dmx-1000. *Computer Music Journal*, 9(1):29–38, 1985.

[115] George Tzanetakis. Marsyas-0.2: a case study in implementing music information retrieval systems. *Intelligent Music Information Systems. IGI Global*, 14, 2007.

[116] George Tzanetakis and Perry Cook. Multifeature audio segmentation for browsing and annotation. In *Applications of Signal Processing to Audio and Acoustics, 1999 IEEE Workshop on*, pages 103–106. IEEE, 1999.

[117] George Tzanetakis and Perry Cook. Marsyas: A framework for audio analysis. *Organised sound*, 4(03):169–175, 2000.

[118] Horacio Vaggione. Some ontological remarks about music composition process. *Computer music journal*, 25(1):54–61, 2001.

[119] Igor Vatolkin, Wolfgang M. Theimer, and Martin Botteck. AMUSE (advanced music explorer) - A multitool framework for music data analysis. In *ISMIR 2010: Proceedings of the 11th International Society for Music Information Retrieval Conference* [2].

[120] Barry Vercoe. *Csound, www.csounds.com*, 2015. Accessed: 2015-05-13.

[121] Ge Wang. *The ChucK Audio Programming Language: A Strongly-timed and On-the-fly Environ/mentality*. PhD thesis, 2008.

[122] Trevor Wishart. *CDP: Composers Desktop Project, www.unstablesound.net/cdp.html*, 2015. Accessed: 2015-05-13.

[123] Guangyu Xia, Dawen Liang, Roger B. Dannenberg, and Mark J. Harvilla. Segmentation, clustering, and display in a personal audio database for musicians. In *ISMIR 2011: Proceedings of the 12th International Society for Music Information Retrieval Conference* [3].

[124] Rudolf H. Zaripov. *Cybernetics and Music (in Russian)*. Znanie, Moscow, 1963.

[125] Aymeric Zils and François Pachet. Musical mosaicing. In *DAFx International Conference on Digital Audio Effects*. DAFx.

[126] Eberhard Zwicker. Subdivision of the audible frequency range into critical bands (frequenzgruppen). *The Journal of the Acoustical Society of America*, (33 (2)):248, 1961.