Dynamic Music Representations for Real-Time Performance: From Sound to Symbol and Back

Grigore Burloiu

"POLITEHNICA" UNIVERSITY OF BUCHAREST

ETTI-B DOCTORAL SCHOOL

Decision No. ..... / .....

Dynamic Music Representations for Real-Time Performance: From Sound to Symbol and Back

Reprezentări Muzicale Dinamice pentru Interpretare în Timp Real

by **Grigore Burloiu**

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
in Electronics and Telecomunications

DISSERTATION COMMITTEE

President	**Prof. Dr. Ing. Gheorghe Brezeanu**	from	**Univ. Politehnica Bucureşti**
Advisor	**Prof. Dr. Ing. George Ștefan**	from	**Univ. Politehnica Bucureşti**
Reviewer	**Dr. Arshia Cont**	from	**INRIA/CNRS/IRCAM Paris**
Reviewer	**Prof. Dr. Ing. Cristian Negrescu**	from	**Univ. Politehnica Bucureşti**
Reviewer	**Prof. Dr. Mircea Florian**	from	**UNATC "I.L. Caragiale" Bucureşti**

Bucharest 2016

Abstract

This thesis contributes to several facets of real-time computer music, using the construct of *dynamic music representations* as a platform towards the online retrieval and processing of information from sound and its (re)interpretation into visual and auditory display.

We design and implement a real-time performance tracker using the audio-to-audio alignment of acoustic input with a pre-existing reference track, without any symbolic score information. We extend this system with a robust tempo modelling engine, that adaptively switches its source between the alignment path and a beat tracking module.

Through several case studies, we contrast this system with the audio-to-score following paradigm, as instanced by the existing state-of-the-art *Antescofo* software. We describe our contributions to the user interface for the programming and execution of *Antescofo* scores, focussing on *visual models* for the coherent notation of dynamic processes. We also investigate avenues for automated generation of interactive music scores by way of formal grammars.

Finally, we harness the potential of musical sound for data exploration and performance mediation, by implementing an *EEG sonification* system as part of an interactive theatre piece.

Throughout, our objective is to increase the capacity of software-based systems for dynamic interaction, which enables more immediate and expressive modes of human-computer music performance.

Contents

List of Figures

List of Tables

Abbreviations

API	**A**pplication **P**rogramming **I**nterface
CQT	**C**onstant **Q** **T**ransform
DFT	**D**irect **F**ourier **T**ransform
DSP	**D**igital **S**ignal **P**rocessing
DTW	**D**ynamic **T**ime **W**arping
EEG	**E**lectro**e**ncephalo**g**ram
FFT	**F**ast **F**ourier **T**ransform
GUI	**G**raphical **U**ser **I**nterface
HMM	**H**idden **M**arkov **M**odel
IMS	**I**nteractive **M**usic **S**ystem
MIR	**M**usic **I**nformation **R**etrieval
MFCC	**M**el-**F**requency **C**epstrum **C**oefficient
OSC	**O**pen **S**ound **C**ontrol
SDK	**S**oftware **D**evelopment **K**it
STFT	**S**hort-**T**ime **F**ourier **T**ransform

Chapter 1

Introduction

Computers have become an established part of our conception and practice of music. Gone are the days when laptops or electronics on stage would be an exotic appearance—today, digital machines that passively treat sound are commonplace, while reactive algorithms and even active feedback loops are becoming more widespread.

A number of challenges in the pursuit of full machine musicianship have stemmed from the essential difference between the ways machines and humans process music information: discrete symbolic computation versus continuous embodied experience.[1] To bridge the gap between the inherent semiotic nature of computers and the dynamical aspect of music, we put forward the phrase "dynamic music representations" to bring together four broadly defined activities:

1. the retrieval of information from music audio signals;
2. the representation of human musical reasoning through computers;
3. the synchronisation and coordination of the above two;
4. the (re)interpretation of computer data into musical sound.

The flexibility of human natural language enables us to describe each of these four as acts of dynamic music representation. We argue that, more than just verbal gymnastics, this construct has the value of connecting a set of perspectives towards human-computer musical interaction.

[1] There are computational views of humans' perception of sound, mainly related to the spectral analysis abilities of the human ear [184]. However, here we refer specifically to the cognition of *music*, which, along with Stockhausen, Leman, Roads and Magnusson cited below, we approach as a dynamic, embodied phenomenon.

This thesis is conceived as a stepping stone made up of contributions to the four modes of dynamic music representation listed above. Far from an exhaustive treatment of the field, we propose a set of specific software building blocks to advance the state of the art.

We identify two main threads underpinning the course of this text. The first traces the pathways between sounds and symbolic representations. In human culture as well as inside computers, these pathways are traversed by *information*. To explain, we start by examining sound and information.

1.1 Sound and Information

Sound is mechanical vibration in a material medium [177]; the study of this physical phenomenon is acoustics. When certain material conditions are fulfilled, sound also affects living beings via the sense of hearing; this phenomenon is the subject of psychoacoustics.

Leman [120] traces the distinction between these two sciences back to the Cartesian mind-body dualism, which lay in opposition to the Greek philosophers' connection between sound and sense through *mimesis*. Both these views can be recognised in modern music cognition research.

Information is also a multi-faceted concept. In computer science, Claude Shannon defined the properties of discrete sources of information [194], which paved the way for the statistical structuring of quantized audio signals, a cornerstone element of computational musicology [145] and computer music [103]. Systems theory restricts Shannon's definition of information, by distinguishing it from raw data: information is data with *meaning* attached [1]. Pierre Schaeffer [39] brought a phenomenological approach to the limits of what information can be interpreted from sound, as mediated by technology.

Turning to the human brain, the nature of information is itself elusive. Mainstream cognitive science [86] treats the brain as a processor of encapsulated information, loosely analogous to how computers process Shannon information. Alternatives include the ecological approach [92] adopted by radical embodied cognitive science [38], which argues that cognition is a direct and embodied process, in which the brain is just one link

in the chain. Indeed, composers and musicologists have often assumed this embodied perspective [120], as seen in this account by composer Karlheinz Stockhausen [203]:

> Sound waves penetrate very deep into the molecular and atomic layers of our selves. Whenever we hear sounds, we are changed. We are no longer the same. ... This is more the case when we hear organised sounds, sounds organised by a human being: music.
>
> (quoted in [178])

Back to computer science, the matter is more objective: information is computed from raw sound data in the act of *representation*, which attaches it to signals and symbols, be they machine-readable or human-readable.

Conversely, when sound is used to (re)interpret information (and we would argue that in music, this is always the case), it gains a representational quality. We look at representations and their dual nature in section 1.1.1 below.

As we have indicated, both processes outlined above are far from trivial. The matter of extracting information from musical audio input has been the subject of the *music information retrieval* (MIR) research domain, working towards convincing *machine listening* [187] or *audio content analysis* techniques [121]. The MIR tasks most relevant to this thesis are:

- score following, the matching of human input to a stored reference [155];
- beat tracking, the identification of a regular pulse in audio [182];
- tempo tracking, emulating the perception of speed of a piece [36];
- harmonic analysis, including the extraction of perceptually [65] and cognitively [200] motivated descriptors.

More specifically, we are concerned mainly with the *online* flavours of these processes, which facilitate "live" interaction and performance. Online content analysis necessitates the information to be extracted in real time, with minimal latency.

Meanwhile, the computational strategies for synthesizing information into sound and music are just as varied, and fall under the general domain of *computer music* [66, 176]. We list the research areas most relevant to our current purposes:

- interactive music, or active machine musicianship [187];
- automatic accompaniment, the generation of a "backing track" that synchronises with a human musician [171];
- sonification, the interpretation and illustration of data through sound [116].

A secondary thread to feature in this thesis is a focus on the *dynamic* aspects of musical processes and their representation. We establish our computationally-oriented understanding of dynamics in section 1.1.2 below, where we also review other notions of dynamics that are involved in sound and music computing.

1.1.1 Representations

The concept of representation in musicology and computer science stems from the domain of semiotics, which is the study of meaning through signs [193]. Representations are semiotic constructs denoting the act of living beings using signs to know the world. We talk of visual representations, digital representations etc to signify the acquiring of knowledge in the form specific to the respective medium.

In digital signal processing, the discretization of an analogue source allows the representation of sound as a series of numeric PCM values [196]. Roads [179], citing Sebeok [192], describes such a representation as *iconic*, in that its formulation mirrors the phenomenon it represents: the sequence of values follows the shape of the original acoustic signal. Through classifier models, we can reach *symbolic* representations. These act as tags, denoting high-level information—one example would be automatic chord detection [200]. We take a brief look at digital signal processing for real-time MIR in section 2.3.1.

Iconic and symbolic representations both play a role in musical scores. For instance, in standard practice Western notation, the correlation between pitch and height on a staff is direct, hence iconic, while the association of note heads and stems to duration is a result of convention, hence symbolic. Both types of representation also appear in the automatic analysis [147] and generation [98] of music structures, and in the design of visual music notation and editing software [30]. This dualism underlies our treatment of computer music notation (section 2.2 and chapter 3) and interaction design (section 4.2), and our experiment with formal grammars in section 4.3.

In electroacoustic musicology, representation denotes sounds that evoke non-sounding qualities:[2] an extrinsic relationship [220]. Let us not stop there. More universally, music is *intrinsically* representational due to its narrative potential [178]: its unfolding in time produces at the very least iconic representations, shaped by the audience's memory and anticipation [109] of their listening experience. This principle of narrative illustration propels the practice of sonification, which we review in section 2.4 and contribute to in chapter 5.

Valuable strategies of discovery in machine musicianship research have stemmed from theories of music cognition and cognitive science [188]. Given the ubiquity of representations in information theory, signal processing, formal languages etc, it might seem odd that their status in human cognition is still up for debate. We mentioned theories of embodied cognition above, preempting the question: do mental representations exist? Are they formed outside of the mind, or at all? We do not attempt an answer; for our purposes, representations are useful in so far as they explain and predict phenomena.[3] Several aspects of cognition can be explained, *sans* representations, as direct sensorimotor interaction and modelled through dynamic systems theory [13, 207]. These include musically interesting phenomena such as beat induction and social coordination—which leads us to examine the role of dynamics in machine musicianship and interaction.

1.1.2 Dynamics

Dynamical systems are mathematical models that describe the time dependence of a point in geometrical space [15]. Generally this involves describing the behaviour of a system using differential equations. Examples in computer music research include beat tracking with a nonlinear oscillator [118] and modeling tempo and duration by coupled oscillators [50]. We add our contribution, a tempo model based on dynamic time warping and control theory [87], in section 4.1.3.

However, in this thesis we employ a wider notion of dynamics, based on its computer science sense, signifying *variability from one program execution to another*. Implementations of dynamical system models necessarily fall under this category, but so do any programs with states linked to environmental or stochastic conditions.

[2] see `http://ears.pierrecouprie.fr/spip.php?article215` .

[3] "God doesn't need to be 'real' to explain church-going, no more than Rules/Norms do to explain rule-following." [12].

Computational dynamics are valuable instruments for machine musicianship. To offer just one cutting-edge example, we cite Eduardo Miranda's exploration of biocomputing, which harnesses the nonlinear properties of single-cell organisms embedded in computer systems to process and reproduce music in naturally non-deterministic ways [143].

We also want to suggest a hierarchy of degrees of dynamic*ness*: loosely speaking, a basic score follower is less dynamic than a score follower plus random cues, which is less dynamic than a score follower plus spectrally triggered cues, which finally is less dynamic than a score follower plus complex reactive model.

This understanding of dynamics informs our whole text; it is essential to our definition of interactive music systems in section 2.1, and to our visualisation models for notating dynamic music processes in chapter 3.

In the ultimate sense, sound and music are dynamic by definition. Both acoustically and psychoacoustically, a music event can never be identically replicated [75]. We do not refer to this implicit dynamic nature of music in our text, but we cannot and would not avoid its presence.

Finally, we must specify that we never use the term "dynamics" here in its music theory sense, denoting the strength with which a sound is to be produced. This is to eliminate any confusion with our designated, computational, sense.[4]

1.1.3 Dynamic Representations / Dynamics Representation

We have outlined how information travels from sound to symbol and back by way of representation, and how dynamic processing of signals and symbols can inject musical life into machines. In turn, the configuration of dynamic processes can be revealed to humans by symbolic representation, enhancing interactivity—especially in live performance.

Leman [120] calls for a subject-centred approach to music computing, implying top-down, multi-modal, subjective, action- and user-oriented strategies. The following contributions of this thesis can be associated with this perspective:

- user-oriented design of a visual interface for computer music authoring and performance (chapter 3);

[4] To be fair to music theory, their "dynamics" is probably the closest to the Greek root of the word, meaning "power".

- perceptually motivated evaluation of tempo modelling (section 4.1.4);
- sonification of modelled subjective experience (chapter 5).

Music software is a *semiotic tool*. Magnusson [129] writes that humanity has "gone through three stages in their development of tools: tools proper (flints, hammers, flutes, guitars), machines (mills, cars, mechanical music machines, pianolas), and automated, informational, or cybernetic machines (computers)." With this in mind, we work towards transferring the immense symbolic power of software into musical, hence embodied, relationships.

In our culture, the form taken by symbolic representations is not necessarily graphical or digital. Gesture also serves to represent musical and cognitive processes: beyond the classical role of the orchestra conductor, we can cite more elaborate frameworks such as *Conduction*[5], a form of guided improvisation where a composer/conductor chooses from a set of directives to trigger and control musical processes in the orchestra in real time[138]. This practice, preceded by the work of musicians like Charles Moffett, Frank Zappa or Sun Ra in the avant-garde jazz and rock scenes, involves a feedback loop where musicians interpret symbolic gestural commands as instrumental actions and relationships, which in turn can be stored into mental "memory banks" and recalled at a later date. All the while, the interpretation is mediated by the orchestra members' individual and group-learned musicianship.

Establishing similar relationships in the computer music realm is one of the main motivations of this thesis. While computers have the advantage of superior storage and recall of parameters and algorithmic patterns, the rich musical abilities of the human musician (or sometimes the average listener) are one of the most difficult emulation challenges for machine intelligence.

In the interests of clarity, we conclude this section with a number of definitions to be used moving forward.

[5] see `http://www.conduction.us` .

Acoustic / electronic scores

We describe by *acoustic input* any audio signal that act as an input into a computer or electronic system. This includes any acoustic source as picked up by a microphone in a musical setting, or any direct audio signal produced by an electric or electronic instrument. The *acoustic score* describes the acoustic input planned for a piece.

Conversely, *electronic input* is any control signal that can enter an electronic instrument or computer. This includes analogue control voltage, mouse movements, random number generators etc. The *electronic score* describes the actions and processes to be executed by the machine, potentially but not necessarily applied to the acoustic and electronic inputs received.

Since there are several inspired taxonomies of electronic and computer music available [47, 103, 178], we are satisfied with the above delimitation of acoustic and electronic scores, which also implies how they can be combined to create *augmented scores* and *electroacoustic* music: all they require are an analogue sound source and electronic manipulation.

Delay / echo

In this thesis we use the word "delay" strictly to refer to translations in time and their associated durations, which can have a positive or negative value depending on the movement taking place forwards or backwards in time.

To avoid confusion with the common *delay audio effect* [177], we will exclusively refer to the latter as an "echo" effect, putting aside the distinctions between echo and delay effects in music production. We can still talk about the *delay time* of an echo effect, referring to the duration of the main repeat interval produced by the process.

1.2 Thesis Objectives

The overarching aim of this thesis is to advance the state of the art in real-time, dynamic computer music in order to enable more immediate and expressive live performance situations. To this end we set out a number of targets, each contributing to an ecosystem of modular interactive music software.

We propose the following five main objectives:

1. To develop new models of visual representation for dynamic computer music. These will support the creation and performance of advanced modes of human-computer musical interaction.
2. To develop score-agnostic musical tracking algorithms that allow the robust synchronisation of a performance in real time directly from the audio, without symbolic information. These will enable musicians to bypass the symbolic interpretation of their material and directly relate to the audio.
3. To qualitatively compare the application of these algorithms with the symbolic model of score following via descriptive interaction scenarios.
4. To investigate possibilities for automatic generation of interactive music scores.
5. To explore the potential of data sonification to enhance artist-audience connection.

1.3 Thesis Outline

The subtitle of this thesis "from sound to symbol and back" roughly describes the sequencing of our text. The work is split into the following chapters:

Chapter 2. Background

A theoretical and technological review of our context is provided. Dynamic computer music is explored, with a focus on interactive music systems (IMS), which couple humans and machines in real-time musical performance. A historical and technical perspective is offered on the issues of visualisation and notation particular to this music. The related problems of score following and tempo tracking are then examined, with a closer treatment of the online dynamic time warping (DTW) algorithm for audio-to-audio alignment. At the border of computer music, the interdisciplinary field of sonification is described, along with its real-time application to electroencephalograms (EEG).

We conclude with a review of the existing infrastructure, showcasing the technology powering these practices, with particular attention given to the platforms featured in our work: Max and *Antescofo*.

Chapter 3. Information Visualisation of Dynamic Sound Processes

A visual notation framework is presented for real-time, score-driven computer music where human musicians play together with electronic processes, mediated by the *Antescofo* reactive software. This framework approaches the composition and performance of mixed music by displaying several perspectives on the score's contents. Our particular focus is on dynamic computer actions, whose parameters are calculated at run-time. For their visualization, four models are introduced: an extended *action view*, a staff-based *simulation trace*, a tree-based *hierarchical display* of the score code, and an out-of-time *inspector* panel. Each model is illustrated in code samples and case studies from actual scores.

Chapter 4. Real-time Coordination of Sonic and Symbolic Material

The first part of this chapter deals with the specific scenario of real-time performance following for automatic accompaniment, where a relative tempo value is derived from the deviation between a live target performance and a stored reference, to drive the playback speed of an accompaniment track. A system which combines an online alignment component with a beat tracker is introduced. The former aligns the target performance to the reference without resorting to any symbolic information. The latter utilises the beat positions detected in the accompaniment, reference and target tracks to (1) improve the robustness of the alignment-based tempo model and (2) take over the tempo computation in segments when the alignment error is likely high. While other systems exist that handle structural deviations and mistakes in a performance, the portions of time where the aligner is attempting to find the correct hypothesis can produce erratic tempo values. The proposed system, publicly available as a Max external object called rvdtw~ , addresses this problem.

The chapter continues with three applications of the rvdtw~ and *Antescofo* systems, evidencing the practical differences between audio-to-audio and audio-to-symbol following, and the possibilities for dynamic extension afforded by the Max environment and the *Antescofo* score language and visualisation system.

Finally, an exploration of formal grammars as a possible tool to automate the process of authoring coherent acoustic-electronic mixed music scores concludes this chapter.

Chapter 5. Data Sonification

This chapter approaches the field of EEG signal sonification from the perspective of multi-modal theatre. A performance setup is described where an actor and an audience member both modulate a pre-designed generative soundtrack in real time. The dramatic and cognitive implications are studied across several performances.

Chapter 6. Conclusions

This thesis brings together several perspectives and approaches on the theory and practice of computer music. In this chapter the different objectives and research avenues are revisited and correlated. Future work is proposed for each strand, along the common theme of dynamic music representation.

Chapter 2

Background

In this thesis we use the construct of dynamic music representations as a guide towards developing software tools for enhancing live human-computer music performance.

This chapter covers essential concepts and related research. We begin with a hierarchy of *interactive music* as a subset of *mixed music*, which in turn is a subset of computer music. We then note interpretations and critiques of the basic definition of interactive music and propose an alternative based on the criterion of coupled dynamics. Next we review the state of the art in mixed music notation and synchronisation, before turning to the area of data sonification.

Finally we discuss the technology infrastructure existing in the field, with a particular focus on the software employed in this thesis: Max (used in chapters 3, 4, 5), Max SDK / C++ (chapter 4), *Antescofo* (chapters 3, 4) and *AscoGraph* (chapters 3, 4).

2.1 Interactive Music Systems

Magnusson [130] explains digital musical applications as *epistemic tools*, extensions of the mind that tie it to the environment. If our cognition extends to external objects and artefacts, then real-time music software acts as a conduit from our musical intention and behaviour into the environment (from designing interfaces and instruments to defining the broader music culture) and *vice versa*, our mind (as the locus of virtuosity, expectation, taste etc) being in turn shaped by our software tools.

This view informs our definition of *dynamic* computer music. We first take a brief look at the history and current state of computer music, before exploring how dynamic computation enables mutuality and complexity in human-computer music systems.

A Short History of Computer Music

The history of computer music dates back to the 1950s, where the term was used to describe works composed with the aid of computers. In the following decades, as technology evolved, the term computer music grew to its modern signification: music where computers facilitate five basic functions: *programming* (composing) of musical parameters, their *storage and retrieval*, the audio *synthesis* of their execution, and the *editing* and *playback* of the resulting material [103]. In the 1960s these were all separate, sequential stages—today, as we will show, they often concur and overlap.

In the 1970s and into the 80s, the arrival of the microprocessor brought real-time digital audio sampling, processing and synthesis abilities first to dedicated hardware units (like the *Synclavier I* (1975), the *Fairlight CMI* (1979) and the Yamaha *DX-7* (1983) digital synthesizers) and in large electronic music studios (such as the Systems Concepts Digital Synthesizer (1977) at CCRMA in Stanford and the *4A*, *4B*, *4C* and *4X Digital Sound Processor* (1974-76) at IRCAM in Paris), and later to the general public with the first consumer-level musical software (Vercoe's *Csound* programming language (1985), the *Performer* early MIDI [1] sequencer (1985) and Puckette's *Max* (1988) graphical development environment).

In the 1990s, the accessibility of real-time sound processing gave birth to various communities and software environments like *Pd* [161], *Max/MSP* [59] and *SuperCollider* [140], enabling *interactive music* situations between performers and computer processes, in the studio and on stage [187]. In parallel, computer-assisted composition tools have evolved to enrich data representation for composers interested in processing data offline to produce scores or orchestration, such as *OpenMusic* [11] and *Orchids* [152].

Today, these and other tools enable various integrated approaches to computer music, some of which we shall now describe.

[1] MIDI: Musical Instrument Digital Interface, a standard protocol that enables symbolic communication of discrete music data between devices.

Four ways to do computer music

We distinguish between *fixed-media* computer music (sometimes called tape, acousmatic, or electronic music [178]), and *mixed music*, where human musicians are paired with computer processes or electronic equipment [47, 135]. Interactive music is a subset of mixed music, where an element of *machine musicianship* [187, 188] allows the computer to act as a performance partner, with each side influencing (and potentially anticipating) the behaviour of the other.

Fixed-media computer music has a long history, from the first examples of tape music and *musique concrète* in the '40s and '50s (pioneered by composers like Halim El-Dabh, John Cage, Pierre Schaeffer et al. [103]), through the early pure electronic music of the '50s [81] and Karlheinz Stockhausen's electroacoustic *Gesang der Jünglinge* of 1955–56 [117] to modern computer-based mixtures of digitally processed synthesized and sampled sounds [176]. A foremost example of the modern fixed-media composer and theoretician is Curtis Roads [178]. His is very much a performative art, but the performance takes place in the studio at material generation phase, rather than the stage presentation. Once the piece is realised, the artist's role in the performance is closer to that of a front-of-house (FOH) engineer, controlling the parameters for the piece's diffusion, i.e. making sure the music is optimally reproduced in the respective environment.

A third strain of computer music that borders on both electroacoustic and mixed music is *live coding* [48], where all the sound is generated by computers and electronics, but the human programmer is at the centre of the musical act, instigating and controlling all the electronic processes in real time. In the past decade, the domain has flourished significantly, with dedicated software libraries [131, 180], notation paradigms [132], online communities[2], and an international conference[3].

A fourth facet of computer music is data *sonification* [100]. We review this area in more detail in section 2.4 below.

Naturally, the above list is not exhaustive: there are ways to musically interact with computers other than machine musicianship, code, and data streams. Indeed, the study

[2] e.g. TOPLAP; see `http://toplap.org` .
[3] the ICLC; see `http://iclc.livecodenetwork.org` .

of new interfaces for musical expression[4] forms an entire research domain. All of them have in common a reliance on music representations to transcode information from sound to machine symbols, and/or from other media to sound.

Defining IMS

Following our definition of interactive music, *interactive music systems* (IMS) are computer and/or electronics-based systems who perform mixed music together with human performers. Rowe [187] defines them as systems "whose behaviour changes in response to musical input." As such, they engage in (at least) two broad processes, equivalent to a human musician's *listening* and *action*: *machine listening* and *live electronics*.

Machine listening implies the real-time analysis of an symbolic or audio musical signal. If this analysis takes place relative to a stored musical reference, Rowe calls the system *score-driven* [187]. Otherwise, the system is *performance-driven*. We adopt this taxonomy, with the caveat that in literal terms, all IMS are performance-driven in the last instance.

We use the term live electronics to cover the response phase of the IMS, consisting in the production of sound by using the musician's input as a source of material and/or as a control interface. There is a long tradition of live electronic music, starting in pre-computer era with the experimental performances of Robert Ashley and Gordon Mumma, the collaborations of Cage and Merce Cunningham's dance company (including pianist David Tudor), Stockhausen's *Mikrophonie* pieces and many others [64, 103, 117]. These practices are carried forward into the present by the rich live electronic (mostly improvisation-based) music scenes of Berlin, Tokyo, New York City, London and elsewhere.

There have been several approaches at fleshing out the specifications of musically useful IMS. Dannenberg introduced the concept of Human Computer Music Performance (HCMP) systems [62], whose maximal architecture is represented in figure 2.1. He notes the limitations of score-driven IMS when faced with "popular music" styles, where synchronisation is more often based on beats than a score, and players are free to more flexibly interpret the music than in score-based music situations. His proposals reach beyond Rowe's IMS definition towards multi-modal input and output representations that facilitate a richer communication between human and machine.

[4] see `http://www.nime.org` .

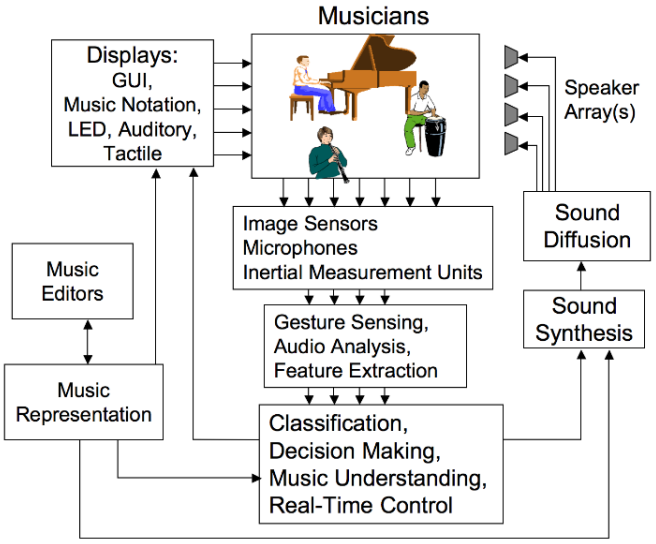


FIGURE 2.1: Overview diagram of a HCMP system (reproduced from [62]).

We also cite the landmark study by Laurie Spiegel [198] which provides an open list of 16-plus axes along which IMS may be classified, including the degree of human participation, amount of practice required for human interaction, number of variables manipulable in real time etc. Various other attempts have been made to organise and understand IMS [72, 219]; we refer the reader to [111] for a more detailed review. For now, we want to take a closer look at the debate over the *interactive* aspect of these systems.

Interaction critiques of the minimal IMS definition

There have been several claims of the insufficiency of Rowe's definition. They argue for a stricter specification of concept of interaction, so as to exclude trivial examples like the Yamaha DX7 from the category of IMS. From the perspective of the computer music craftsman, Jordà [111] and Stark [199] have stressed the importance of *mutual influence* and *relationship complexity* between human and machine.

From the compositional perspective, we can cite the insights of Marco Stroppa [204], Jean-Claude Risset [173] and Joel Chadabe [37]. In his work, Stroppa asks the musical question of juxtaposing multiple scales of time or media during the composition phase, and their evaluation during performance. He further remarks on the poverty of musical expressivity in then state-of-the-art real-time computer music environments as opposed to computer-assisted composition systems or sound synthesis software. Risset takes this one step further, arguing that interactive music systems are less relevant for composition than performance. Finally, Chadabe questions the very use of the term "interaction"

as opposed to "reaction". Effectively, many such systems can be seen as *reactive systems* (computers reacting to musicians' input), whereas interaction is a two-way process involving both specific computing and cognitive processes.

A computational approach to the interaction dilemma

To address the above criticisms and to define the current state of the art in computing terms, we propose to define *dynamic* mixed music, and implicitly dynamic computer music, following the concepts established in section 1.1.2.

In this paradigm, IMS and their surrounding environments (including human performers) are integral parts of the same system and there is a feedback loop between their behaviours. An ubiquitous example of such dynamics is the act of collective music interpretation in all existing music cultures, where synchronisation strategies are at work between participants, in a phenomenon known as *entrainment* [43]. Dynamic computer music design extends this idea by defining processes for composed or improvised works, whose form and structure are deterministic on a large global scale but whose local values and behaviour depend mostly on the interaction between system components—including human performers and computers/electronics.

The two-way interaction standard in [37] is harder to certify when dealing with cognitive feedback between machine generated action and the human musician, and many of the programming patterns might be strictly described as *reactive* computing. For this reason we argue that the computationally *dynamic* aspect of the electronics should be their defining trait, as this in turn implies a certain degree of mutualism and complexity in the system.

In addition to the above, particular attention might be afforded to different notions of *time* (symbolic/musical/beat and physical/actualised/ms[5]) and their online realisation via joint human-computer action. This type of temporal coupling has been enabled by synchronous programming languages such as *Esterel* [18], and is a motivating factor in our perspective on modern IMS [55].

As such, we attempt an updated definition: IMS are systems whose behaviour *is dynamically coupled* to musical input.

[5]We further touch on the distinct notions of time in section 2.5.5 and chapter 3.

Our formulation might be of limited applicability outside this thesis, as it relies on our concept of dynamics from section 1.1.2. Moreover, it presents a degree of ambiguity: must the dynamic computing pertain to the input analysis or to the response? We hesitate to give a definitive answer here, except to suggest that dynamic input analysis is necessary but not sufficient. Furthermore, our claim that stronger dynamics generally correlate with a higher degree of interactivity is for now just an intuition, and has yet to be formally validated. In the end, our stipulation of dynamic coupling is no guarantee of quality—there remain the dangers of bad design.

2.2 Mixed Music Notation

We now approach the issue of notation for the authoring and performance of mixed music, which is a central concern particularly in score-driven IMS design.

Composer Philippe Manoury has theorised a framework for mixed music [135], introducing the concept of *virtual scores* as scenarios where the musical parameters are defined beforehand, but their sonic realisation is a function of live performance. One example is the authoring of music sequences in beat-time and relative to human performer's tempo; another example is the employment of generative algorithms that depend on the analysis of an incoming signal (see [136] for an analytical study).

A uniting factor in the diversity of approaches to mixed music can be found in the necessity for *notation* or *transcription* of the musical work itself. A survey on musical representations [218] identified three major roles for notation: recording, analysis and generation. We would specifically add performance to this list. Despite advances in sound and music computing, there is as yet no fully integrated way for composers and musicians to describe their musical processes in notations that include both *compositional* and *performative* aspects of computer music, across the offline and real-time domains [163], although steps in this direction can be seen in the OSSIA framework for the i-score system [35].

To facilitate the understanding of the field, we distinguish notational strategies for mixed music into three categories, before noting how composers might combine them to reach different goals.

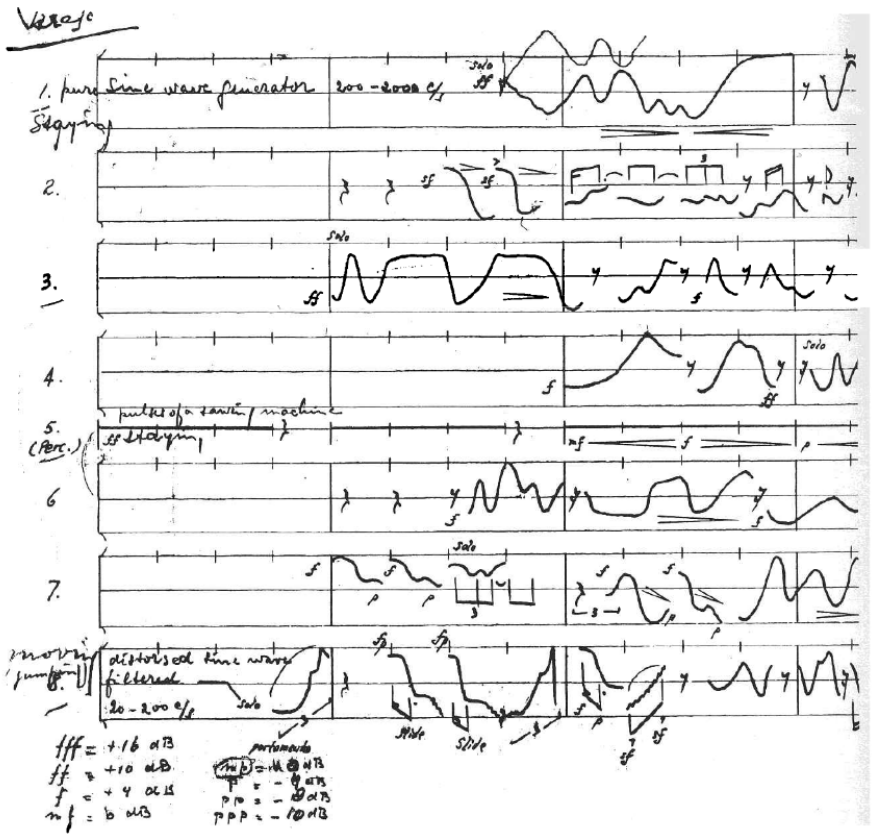


FIGURE 2.2: Excerpt of Varèse's sketch score for *Poème électronique*, published by Philips (reproduced from [178]).

Symbolic graphical notation

Early electroacoustic music scoring methods evolved in tandem with the expansion of notation in the mid-20th century towards alternative uses of text and graphics. Written instructions would specify, in varying degrees of detail, the performance conditions and the behaviours of the musicians. In addition, symbolic graphical representations beyond the traditional Western system could more suggestively describe shapes and contours of musical actions. While these methods can apply to purely acoustic music, the introduction of electronics came without a conventional performative or notational practice, and so opened up the space of possibilities for new symbolic graphical notation strategies.

Major works of this era include Stockhausen's *Elektronische Studie I* and *II* (1953-54), the latter being the first published score of pure electronic music [117], and Edgard Varèse's *Poème électronique*, whose graphical score is excerpted in figure 2.2. These examplified a workflow where the composer, usually aided by an assistant (such as G. M. Koenig at the WDR studio in Cologne), would transcode a manually notated symbolic score into electronic equipment manipulations or, later, computer instructions,

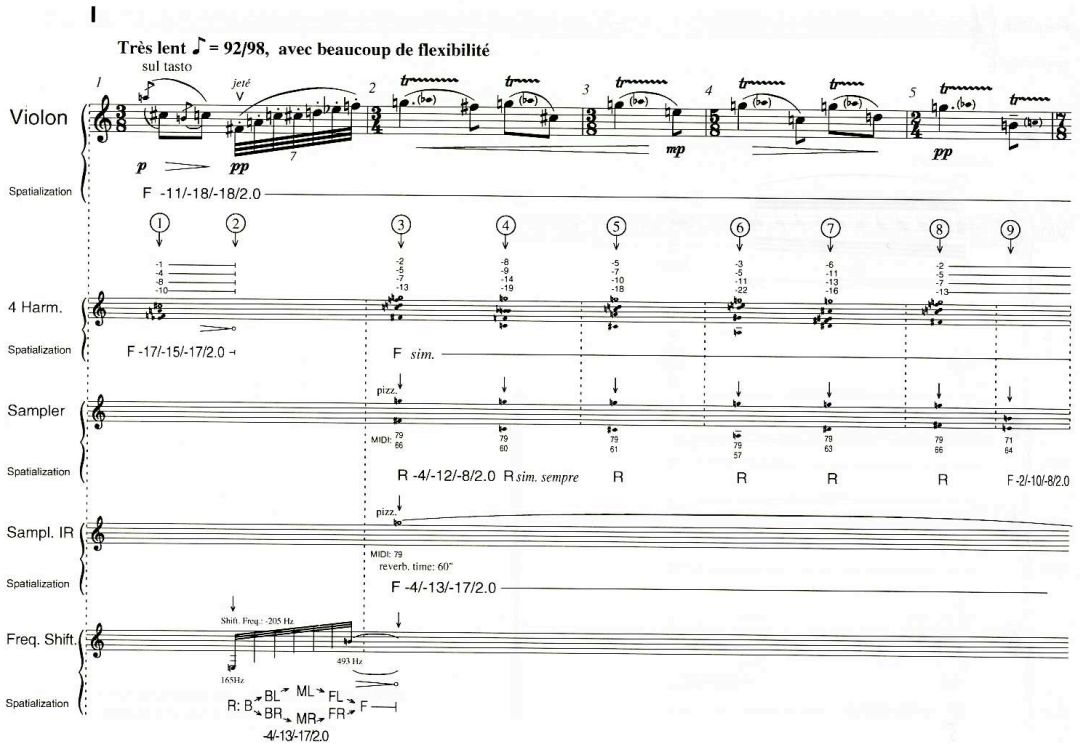


FIGURE 2.3: The first page of the *Anthemes II* score for violin and electronics, published by Universal Edition. The numbered cues indicate electronic action triggerings.

to produce the sonic result. An instance of this paradigm, which continues today, is the collaboration of composer Pierre Boulez and computer music designer Andrew Gerzso for *Anthemes II* (1997), whose score is excerpted in Figure 2.3.

Graphics-computer integration

Before long, composers expressed their need to integrate the notation with the electronics, in order to reach a higher level of expressiveness and immediacy. Iannis Xenakis' UPIC [222] is the seminal example of such a bridging technology: his *Mycenae-α* (1978) was first notated on paper before being painstakingly traced into the UPIC. The impetus to enhance the visual feedback and control of integrated scoring led to systems such as the SSSP tools [30], which provided immediate access to several notation modes and material manipulation methods. Along this line, the advent of real-time audio processing brought about the integrated scores of today, underpinned by computer-based composition/performance environments. Leading examples in the field are the technical documentation databases at IRCAM's Sidney[6] or the Pd Repertory Project[7], hosting self-contained software programs, or *patches*, which can be interpreted by anyone with

[6] `http://brahms.ircam.fr/sidney/`
[7] `http://msp.ucsd.edu/pdrp/latest/files/doc/`

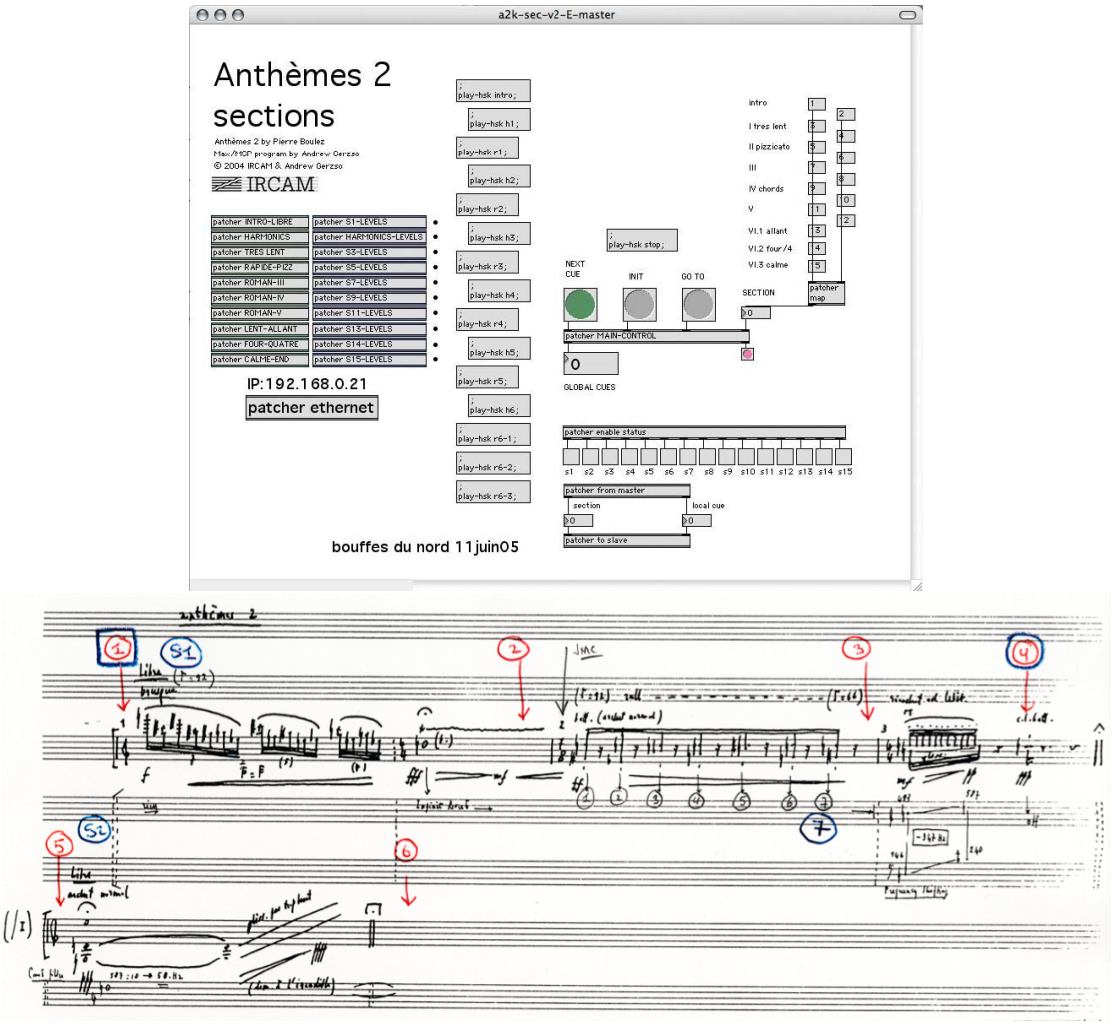


FIGURE 2.4: Top: A portion of the *Anthemes II* Max/MSP patch from 2005. All electronic processes are triggered from a list of cues.
Bottom: Excerpt from the first page of the *Anthemes II* computer music performer score. The marked cues correspond to interaction points with the dedicated patch.

access to the required equipment. These patches serve as both production interfaces and *de facto* notation, as knowledge of the programming environment enables one to "read" them like a score. Since most real-time computer music environments lack a strong musical time authoring component, sequencing is accomplished through tools such as the *qlist* object for Max and Pd [219], the Bach notation library for Max [2], and/or an electronics performer score such as the one for *Anthemes II* pictured in Figure 2.4.

Dynamic notation

Finally, a third category is represented by *dynamic scores* – these are Manoury's virtual scores [135], also known in the literature as interactive scores due to their connection to real-time performance conditions [60]. Interpretations range from prescribed musical

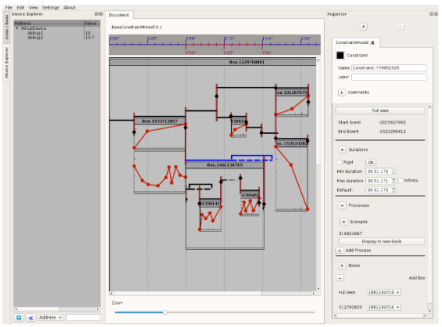


FIGURE 2.5: A prototype of i-score 0.3, featuring conditional branching and dynamic constraints (reproduced from [35]).

actions to "guided improvisations", where the musicians are free to give the notation a personal reading [42], or the electronic algorithms have a degree of nondeterminism. Here, composers are responsible for creating a dynamic roadmap whose acoustic realisation could be radically[8] different from one performance to another.

A question arises: how are such dynamic interactions to be notated? While regular patches can already reach high levels of complexity, the problem of descriptiveness increases exponentially once time and decision-making are treated dynamically. The low-level approach is typified by the Pd software, which was designed to enable access to custom, non-prescriptive data structures for simple visual representation [162]. One step higher is an environment like *INScore*, which provides musically characteristic building blocks while retaining a high level of flexibility through its modular structure and OSC-based API [60]. More structured solutions are provided by OSC[9] sequencers with some dynamic attributes such as IanniX [44] and *i-score* [4]. In particular, *i-score* uses a Hierarchical Time Stream Petri Nets (HTSPN)-based specification model [67], enabling the visualisation of temporal relations and custom interaction points. The dynamic dimension is however fairly limited: an effort to enhance the model with conditional branching concluded that not all durations can be preserved in scores with conditionals or concurrent instances [210]. By replacing the Petri Nets model with a synchronous reactive interpreter, [5] achieved a more general dynamic behaviour, accompanied by a real-time *i-score*-like display of a dynamic score's performance. Still, this performance-oriented visualisation does not display the potential ramifications *before* the actual execution.

[8] (in terms of structure, duration, timbre etc)

[9] OpenSoundControl, a multimedia communication protocol: `http://opensoundcontrol.org/`.

This is generally the case with reactive, animated notation[10] (e.g. Rodrigo Costanzo's *dfscore*[11]): it does a good job of representing the current musical state and relevant instructions, but does not offer a global, out-of-time perspective of the piece. One significant development to the i-score framework enables conditional branching through a node-based formalism [35], as seen in figure 2.5. Meanwhile, more complex structures (loops, recursion etc.) still remain out of reach.

Naturally, much contemporary music makes use of all three types of notation outlined thus far. Composers often mix notational strategies in an effort to reach a two-fold goal: a (fixed) blueprint of the piece, and a (dynamic) representation of the music's indeterminacies. On the one hand, a score might lend itself to analysis and archival [17]; on the other, notation is a tool for composition and rehearsal, which in modern mixed music require a high degree of flexibility. But perhaps most importantly, the score serves as a guide to the musical performance. As such, the nature of the notation has a strong bearing on the relationship of the musician with the material, and on the sonic end result.

2.3 Score Following and Tempo Tracking

As the terminology suggests, score following is an integral part of score-driven IMS, as well as several other kinds of uses such as automatic page-turning [8], performance analysis [69], query by humming [157], or intelligent music editors [63]. Historically, the two main motivations behind real-time score following have been automatic accompaniment and machine musicianship. These two tasks both require that, aside from estimating the current position of a musician in a known score, the system also include a *tempo model*, to control the speed of the accompaniment or the synchronisation of the machine actions to the human input, respectively.

The first steps into score following were taken independently in 1984 by Dannenberg [61] and Vercoe [214]. Their approaches were based on string-matching techniques, aligning the symbolic (MIDI) input data to a stored score string. This paradigm remained in

[10] in the literature, this kind of "live" notation is sometimes called dynamic notation [42], regardless of the underlying scenario being computationally dynamic or not. In this thesis, we use the term "dynamic" with regard to notation for the scoring *of* dynamic music in general. While the notation itself may be dynamic, this is not a necessary condition.

[11] see `http://www.rodrigoconstanzo.com/2015/11/dfscore-2` .

place in the 1990s [165], with the main improvements being related to the support for real-time audio input analysis [160]. At the turn of the century, stochastic methods started being introduced to probabilistically model score states [95], most significantly through hidden markov models (HMM) [32, 154, 155, 170].

The '00s saw a rapid increase of the number and variety of approaches to score following, with the field reaching a mature form. In 2006 it joined the list of MIREX[12] research tasks and an evaluation method was established [53], based on the measurement of temporal alignment error between score markers and live input. Two major systems that grew out of the HMM-based approach, each featuring robust score state and tempo modelling, were Christopher Raphael's *Music Plus One* [171] and Arshia Cont's *Antescofo* [50]. Very roughly speaking, these systems start by codifying the live audio and the score into series of symbolic states, which are then aligned by statistical and machine learning methods.

The converse path was opened by Simon Dixon [69], with the introduction of the online variant of the dynamic time warping (DTW) algorithm. This approach synthesizes the score into an audio rendition (or disregards the symbolic score altogether and treats an audio reference directly) and then aligns the input audio stream to the reference audio.

Modelling *tempo* is a significantly different task from sound marker alignment [52, 71, 106, 139]. While individual onsets are perceived as instantaneous events,[13] tempo is more *explicitly* dynamic, describing the perception of musical speed over time. A good score follower will account for both of these phenomena [50, 171].

The current decade has seen the proliferation of established score following designs in the public sphere, in both the artistic and the commercial educational and gaming domains. Meanwhile, researchers are producing alternative approaches to the audio-to-audio alignment task such as particle filtering [223], as well as strategies that blur the line between the probabilistic and audio alignment paradigms [33, 73, 146].

In our practice, as evidenced by this thesis, we use both performance following paradigms for the appropriate applications. In musical situations that can be easily formalised into sequences of event states, *Antescofo* has proven itself time and again a robust

[12] Music Information Retrieval Evaluation eXchange; see `http://www.music-ir.org/mirex/wiki/MIREX_HOME` .

[13] (whose timings have a dynamic relationship with the corresponding audio waveform peaks [215])

choice. Meanwhile, we also want to explore material that is less susceptible to symbolic description—hence our development of a bespoke online DTW-based performance tracker in section 4.1.

At the most granular level, DTW can be seen a special case of HMM [151], if we look at the frame-to-frame warping process through the prism of state transitions. To clarify this statement, we must first introduce some necessary notions of digital signal processing theory, before exploring the online DTW algorithm in more detail.

2.3.1 Digital Signal Processing Primer

We now run through some essentials of acoustics, DSP and audio content analysis theory. For a more thorough approach we refer the reader to [121, 176, 196, 197].

As we mentioned in section 1.1.1, the digital sampling of sound waves produces a digital *signal*—a sequence of numbers, or *samples* representing the amplitude of the sonic air pressure change at a specific point in time. In the sense that each audio sample describes an instant in time, digital audio signals can be considered *time series*.

Tracing the amplitude of a signal over time produces the amplitude *envelope*; the top part of figure 2.6 displays such an envelope. The positive and negative components correspond to the sequential compression and rarefaction of air molecules caused by a loudspeaker membrane moving back and forth, producing sound energy.

An ideal sonic *sine wave* generator would produce a vibration of air proportional to $\sin 2\pi ft$, where t denotes time and f is the number of cycles per second. f is the *frequency* of the tone (measured in Hz), and $1/f$ is the *period*. A sine wave is a *periodic* tone. True periodic tones do not exist in nature; sounds range from sine-like "pure tones" to random noise, with the vast majority of quasi-periodic tones falling in between, consisting of a multitude of concurring vibrations, each with their own magnitude and frequency.

The human hearing range is conventionally said to cover tones between 20Hz and 20kHz. The *sampling (Nyquist) theorem* states all content up to a frequency of $f_S/2$ in a continuous signal can be fully reconstructed from a digital signal of a f_S sample rate. This means that the audible spectrum can be fully reproduced from signals sampled at 40kHz or higher—in practice, commonly used sample rates are 44.1kHz, 48kHz and above.

Thanks to the *Fourier Transform*, we are able to decompose any quasi-periodic signal as a finite sum of sinusoids. In discrete time, these components can be computed by the *Discrete Fourier Transform* (DFT):

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i\frac{2\pi kn}{N}} \ , \tag{2.1}$$

where $k \in [0, N)$, and N is the length in samples. DFT can be computed in $\mathcal{O}(N^2)$ time; in practice, the Fast Fourier Transform (FFT) is used, which runs in $\mathcal{O}(N \log(N))$ time.

In order to obtain a time-frequency representation, where spectra are computed over a series of short, overlapping *frames*, the *Short-Time Fourier Transform* (STFT) is used:

$$X_k(\tau) = \sum_{n=-\infty}^{\infty} x_n w_{n-\tau} e^{-i\frac{2\pi kn}{N}} \ , \tag{2.2}$$

where w_n is a *window* function and τ is a multiple of the *hop size*, or the number of samples elapsed between windows. By adjusting the window and hop size, we trade off between frequency resolution (large windows) and time resolution (short windows). Large overlap ratios help increase resolution, but decreasing the hop size below a certain value will result in oversampling. Throughout this thesis we use a window size of 2048 samples and a hop size of 512 samples, equivalent to 46.4ms and 11.6ms respectively, which gives the STFT a 21.6Hz frequency resolution at a 44.1kHz sampling rate.

The STFT is a basic example of a *mid-level representation* [82] : it reveals information about a signal which can be leveraged to compute high-level *features*. Other mid-level representations include the *Constant Q Transform* [21] (featuring logarithmically-spaced frequency bins to closer match human perception) and the *Discrete Wavelet Transform* (same, using short bursts instead of sine waves as the basic building blocks).

Feature vectors are multi-dimensional time series of descriptors that provide time-frequency information which is directly useful in MIR algorithms. Figure 2.6 shows different feature representations, where the each vector occupies a vertical stripe on the timeline, and the intensity of the cells corresponds to their magnitude.

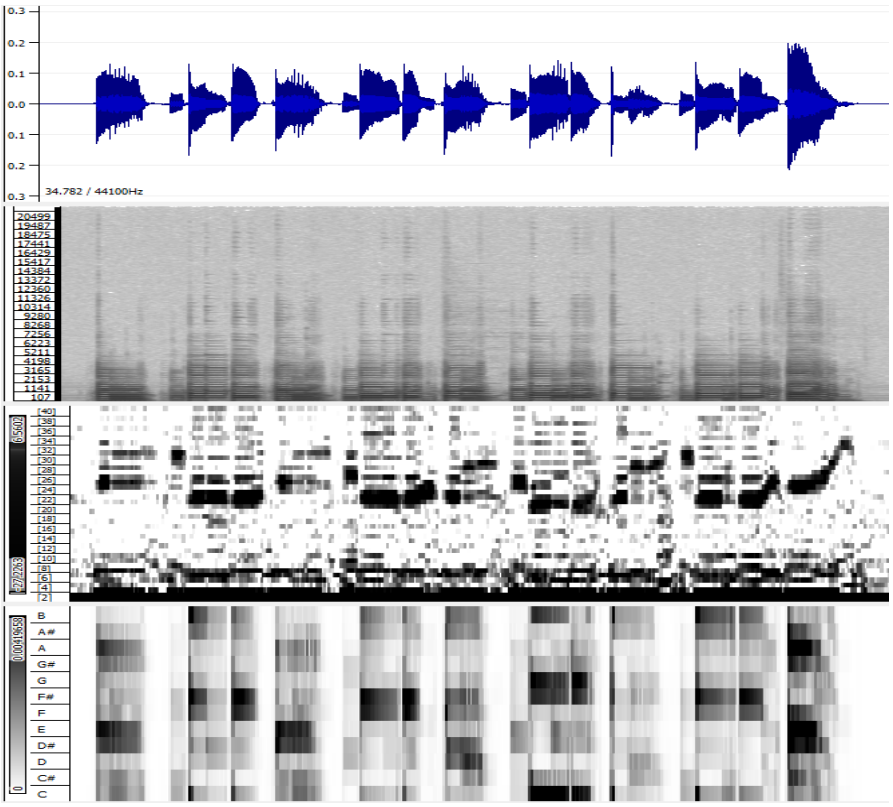


FIGURE 2.6: Time series representations of sound.
Top: direct input electric guitar signal.
Mid-top: STFT spectrogram. Mid-bottom: 40 MFCCs. Bottom: 12-bin chromagram.
2048 sample window, 512 sample hop at 44.1kHz sample rate.

Chromagrams, also called chroma vectors or pitch class profiles, represent the amount of energy in each semitone accumulated across octaves [90]. They are a common choice for tasks such as chord recognition and audio alignment [8, 108, 148, 150, 156, 200]. Improvements to chroma features have been proposed, most notably to add onset information to the existing pitch class data [9, 83], which help improve temporal accuracy—the trade-off being that when soft or repeated onsets that do not appear in the reference are played by a musician, they can be missed or falsely detected, respectively.

Other types of features vectors used in audio alignemnt have so far included pitch histograms [108], spectral energy difference vectors [69], the CQT [70, 167] and Mel-Frequency Cepstrum Coefficients (MFCC) [94, 108], among others.

2.3.2 Audio Alignment with Online DTW

Online DTW is an adaptation of the standard dynamic time warping algorithm, which aims to find the optimal alignment path between two different sequences of numbers or

vectors. This alignment path needs to be monotonous, continuous, and bounded by the ends of both sequences. DTW has seen wide use in tasks such as speech recognition [166] and gesture classification [104], as well as offline audio-to-score alignment [70]. For the real-time audio alignment task, two problems are raised: (1) an incomplete input sequence, since the future input is unknown, and (2) computation complexity, since the system must run in real time. The online DTW algorithm addresses these issues.

We begin with a brief description of the algorithm, as introduced by Dixon [69], before surveying some significant extensions of the basic algorithm from the past decade.

Dixon

Online DTW aligns a *target* time series $X = x_1 \dots x_m$ to a *reference* $Y = y_1 \dots y_n$, where X is partially unknown: only the first t values are known at a certain point. The goal is for each $t = 1 \dots m$ to find the corresponding index h_t, so that the subsequence $y_1 \dots y_{h_t}$ is best aligned[14] to $x_1 \dots x_t$. The alignment path p is a sequence of (t, h_t) tuples connecting the origin with the current position for the lowest global match cost. For audio alignment tasks, x_i and y_j are audio feature vectors, such as the ones mentioned in section 2.3.1.

The distance between vector frames from the two different series is defined by a local cost function $d(i, j)$ which is 0 in case of a perfect match and positive otherwise. We use the Euclidean distance measure to compute the local cost matrix d:

$$d(i,j) = \sqrt{\sum_k (x_{i,k} - y_{j,k})^2} \ . \tag{2.3}$$

It is now possible to define the local match cost $D(i, j)$. Several options have been used in the past, as reviewed in chapter 4 of [166]. We have found the following to be a reliable local match cost measure:

$$D(i,j) = \min \left\{ \begin{array}{l} D(i-1, j-2) + w_{\text{side}} \cdot d(i,j) \\ D(i-1, j-1) + w_{\text{diag}} \cdot d(i,j) \\ D(i-1, j-2) + w_{\text{side}} \cdot d(i,j) \end{array} \right\}, \tag{2.4}$$

[14] In cases where a frame x_t is assigned to several frames in Y, our implementation sets h_t to point to the last of these.

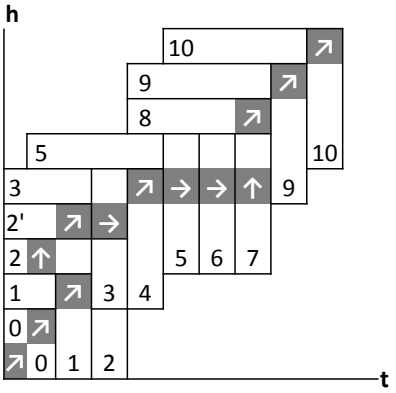


FIGURE 2.7: The classic online DTW algorithm [69]. A minimum match cost on the right border produces a new column, on the top a new row, or both when the minimum is on the corner. Step 7 produces a new row because the `MaxRunCount` for columns has been reached.

where the first two lines and columns of D are ∞, except for $D(0,0) = d(0,0)$ and $D(1,1) = d(0,0) + w_{\text{diag}} \cdot d(1,1)$. w_{diag} and w_{side} are weighting coefficients that help orient the path.

The alignment path is defined as being monotonous and continuous, with a local slope constraint. It proceeds stepwise, with a limited number of `MaxRunCount` consecutive steps in any direction. This instates a tempo boundary: for `MaxRunCount` $= 3$, the local tempo cannot be more than 3 times slower or faster than the reference, which is a good assumption under regular conditions

If ideally the D matrix is of a steadily-increasing size $t \times n$ at any point in time t, in practice it makes sense to save memory and computation time by using a rolling search window of size $c \times c$. For each incoming frame x_t, the previous c steps are known and the path advances by computing a new row, a new column, or both, depending on where the current minimum match cost is found on the search window's frontier – a row, a column or the top-right corner, respectively. The process is exemplified in figure 2.7, for $c = 4$ and `MaxRunCount` $= 3$. This algorithm enforces the monotony and continuousness of the alignment path (it's impossible to decrement the current column or row) and limits the computational burden.

Since 2005, Dixon's synchronisation research output has largely focussed on the offline domain, with the notable exceptions the Windowed Time Warping (a progressive-scale variation of DTW that divides the path into a series of windows) with Robert

Macrae [128] and teaming up with Andreas Arzt and Gerhard Widmer on two papers cited below.

Arzt and Widmer

Together with Dixon, Arzt and Widmer introduced a backward-forward strategy that reconsiders past decisions [8], as part of a score following application designed for page turning. The method periodically computes a smoothed backward path, starting from the current (t, h_t) coordinates, inside a set window in the immediate past. Once the window edge is reached, a new forward path is computed starting from this point, using standard DTW (which is non-causal, so is assumed to be more accurate than the initial online alignment path). If the new path ends on (t, h_t) again, the current position is considered validated; otherwise the algorithm moves along the X or Y axis to meet the new end point. The main effect is a greatly improved tolerance to musician error, since the system now periodically verifies its decision. We introduce a variant of this strategy in section 4.1.1, streamlining it to a single backward DTW trajectory instead of a backward-forward pair.

A second significant contribution of Arzt and Widmer was the "simple" tempo model [6] that uses the smoothed backward path to detect $r = 20$ onsets at least 1 second in the past, and using the slopes of the path rectified between these onsets as relative tempos τ_t. Finally, the current tempo estimate is computed as:

$$\tau = \frac{\sum_{i=1}^{r}(\tau_i \cdot i)}{\sum_{i=1}^{r} i}. \tag{2.5}$$

In section 4.1.3 we propose four different tempo models that serve the same purpose, without using onset information from a symbolic score.

Arzt and Widmer teamed up again with Dixon to propose an improved feature vector representation based on adaptive distance normalisation [9], which they most recently leveraged in a multi-agent model that tracks several reference recordings in parallel for improved robustness [7].

Tracing these developments reveals the continuing relevance of online DTW for real-time performance tracking. Among several competing techniques each with their own

particular strengths, online DTW plays a significant role at the cutting edge of the field [7, 33].

2.4 Sonification

Modern advances in acoustics, psychoacoustics and DSP research and technology, and the growing issue of big data knowledge extraction are increasingly promoting sonification and auditory display as not only an alternative to visual (graphical and symbolic) means of representation, but as the primary, most immediate solution for extracting information from certain complex data. [99]

We mention a few of the characteristics of sound and hearing that shape the properties of sonification as a representation tool:

- sound has an inherently temporal dimension, which naturally orients it towards time series data;
- humans have an exquisite ability to identify specific facets of sound (spatialisation, source detection and separation, speech, music form and structure) which means we can comprehend rich data relationships in real time. Best of all, we learn these abilities and easily train them on varied material;
- the flipside to the above is that the ear needs time to adapt to novelty, which is why system stability is needed in a sonification to make sure listeners can learn to interpret it;
- in certain practical situations humans are (temporarily or permanently) not able to see a visual display, which might be either obscured or overloaded with information. Here sound can act as a representational aid.

Sonification is a heavily interdisciplinary domain. Hermann et al [99] propose the diagram in figure 2.8 to lay out the circuit of information in the sonification cycle, and the associated scientific disciplines. Our present focus is on computer music and data exploration applications. For an overall study of the field, we refer the reader to [100].

Studies from the computer music perspective [221] have underlined the psychoacoustic and cultural considerations that any musical sonification must take into account. For instance, simply assigning dimensions such as pitch and loudness to different data series

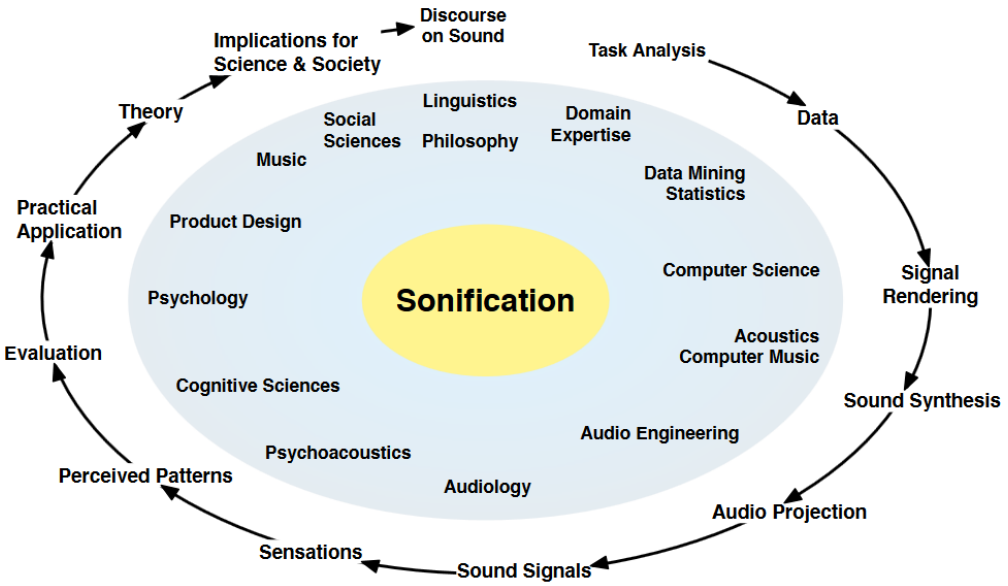


FIGURE 2.8: The "circle of interdisciplinarity" of sonification and auditory display (reproduced from [99]).

can lead to confusing and/or unpleasant combinations. Lack of consideration of auditory perception and attention, and automatic generalisations of visual practices, are common pitfalls in data sonification [85].

One key to successful sonification is the applicability of the source material to the generation technique. The electroencephalogram (EEG), monitoring the electrical activity of the brain's surface, appears particularly fit for the task. Relating back to Roads' comments about the emotional narrative aspect of music from section 1.1.1, it might be the EEG's potential for emotion recognition [77, 125] that attracts musicians and researchers to engage in emotion alchemy.

Brainwaves already have a long history of being used as a source for real-time music generation, starting from the experiments of composer Alvin Lucier and physicist Edmond Dewan in 1964-65, leading to *Music for Solo Performer*, which leveraged the performer's alpha signals to achieve percussive effects [20]. Artistic activity related to electronic brainwave music grew in the 60s and 70s, then fell into a slump due mainly to technological stagnation. Consequently, the 90s onwards have brought about a steady revival of interest in the field.

A modern review of the area of real-time EEG sonification [213] shows a ten-fold increase in the number of publications between 2002 and 2012, indicating a rapid agglomeration of the field. They identify four main application areas for online EEG sonification:

1. *monitoring*, informing about the user's brain state e.g. an anaesthetist during surgery;
2. *neurofeedback*, let the user alter their brain state e.g. for post-stroke rehabilitation;
3. *brain computer interface (BCI) feedback and communication*, including brain controlled musical instruments;
4. *musical compositions*, generated and controlled through brain patterns.

Certainly, some applications can fall under several categories. For instance, an orchestral sonification for brain self-regulation [102] would fall under categories (2) and (4). Musical installations such as [123] stretch the limits of categories (3) and (4). Our proposed project, which we describe in chapter 5, might be classified as (4) with secondary elements of (1) and (3).

De Campo [31] advanced a "design space map" for data exploration sonification, which provides a systematic arrangement of sonification strategies, with specific applications for EEG data. The categories proposed are *continuous* (which includes direct *audification*[15] and *parameter mapping* onto continuous sounds), *discrete point*, and *model-based* data representation. We used the study's taxonomy and EEG-related strategies as a basis for our algorithms in section 5.1.2.

2.5 Existing Technology Infrastructure

> Tools for electronic art-making can be arranged in a spectrum reaching from finished applications to development environments. Any real software tool will lie somewhere in between these extremes. It's easy to suppose that in more creative, exploratory art forms, the artist will have to lean to the "development environment" end of the spectrum.
>
> (Miller Puckette [164])

Now that we have outlined a field of dynamic music representations, we can turn to implementation techniques. We refer here specifically to software, although hardware is of course essential in any interactive system. The tools explored hereon require audio interfacing equipment, and can benefit from MIDI and/or OSC-compatible controllers [176].

[15] Audification is defined as the direct translation of a data waveform into sound [115].

While some have fixed audio-specific functionality, others can be extended with custom visualisation models or external hardware for e.g. lighting, motion tracking or scenography [122].

We look at the practice of computer music software design and some of the tools underlying it, and then study options for the specific task of tracking human music signals, before homing in on a workflow that integrates the above into a fully-featured IMS platform.

2.5.1 Computer Music Design Environments

The phrase computer music design can refer to (at least) two things: in hardware, it is the creation of music controllers and interface paradigms [56]; in software, we mean the programming of systems to realise certain musical scenarios, pieces or classes of pieces [176, 219].

Our present work is hosted by Cycling '74's Max environment[16], which is built around the paradigm of dataflow programming [110]. Its visual nature [101] (as opposed to text-heavy functional programming languages [19]) has made it accessible to artists, audio engineers and analogue synth musicians who are accustomed to thinking of sound processes in terms of states and flows.

As we noted in section 2.2, one of the features that makes Max a powerful language for mapping music interaction is its immediate access to the program structure, state and user interface, meaning that any program change is instantly represented to the user—visually and, once audio signals are involved, sonically. This means that the planning, implementing, debugging and performance situations are now no longer separate phases, but blend into each other as needed.

A basic illustration of Max visual patching is shown in figure 2.9. The basic building block is the *object*, a self-contained algorithm that may receive input (through one or more *inlets*), generate output (through *outlets*), or both. Messages and signals are passed between objects through so-called *patch cords*. These travel at *control rate* (millisecond-level accuracy) and *audio rate* (sample accuracy), respectively.

[16] also known as Max/MSP [59].

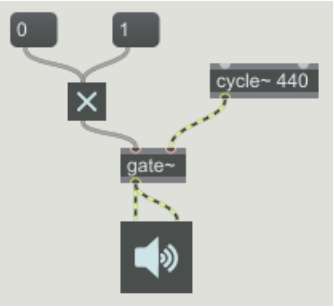


FIGURE 2.9: Dataflow programming in Max 7 (2015). A 440hz sine wave *signal* is passed through a *gate*, which is opened/closed by a *toggle*, whose state is controlled by on/off (`0`/`1`) *messages*. The gate signal output is routed to the computer's physical audio out port for hardware playback.

Another major strength of Max is its interfacing ability, whether through the MIDI and OSC protocols, or through a host of third-party APIs for various external tools. One main example is the extension into the performance-oriented digital audio workstation *Ableton Live* through the dedicated *Max for Live*[17] package, enabling the "remote control" of Live's multi-track sequencing functions from inside Max.

We already mentioned in section 2.1 the existence of several environments with similar capabilities to Max, many of which have evolved in time and are in wide use today (notably Pd, SuperCollider and Csound). To this list we can add the related field of *creative coding*: frameworks such as *Processing*[18], *openFrameworks*[19] and *vvvv*[20] each featuring OSC capabilities. A modern modular system might feature a Max-based score follower triggering generative audio in SuperCollider and a motion graphics-based visualisation in vvvv.

For reasons of familiarity our present work remains within the Max framework, which we augment with external objects when the built-in functionality proves insufficient.

[17] see `https://www.ableton.com/en/live/max-for-live` .
[18] see `https://processing.org` .
[19] see `http://openframeworks.cc` .
[20] see `https://vvvv.org` .

2.5.2 Developing Max Externals in C / C++

Although Max itself is not open software, it offers a powerful C-language API for designing third-party objects, or *externals*[21]. The associated SDK[22] includes examples and documentation.

We will not reproduce the information existing in the official documentation or related materials on the topic [126, 188]; we just present our annotated template that converts the basic C MSP external code into C++, which we found to be more readable and easy to use.

The header and source files in listings 2.1 and 2.2 both start with C API-related admin, while the actual object-specific functionality is encapsulated neatly in a C++ class at the latter part of the files. Once the internal framework is configured, the programmer can forget about it and focus on the C++ code. We used this strategy in developing the rvdtw~ external in section 4.1.

```
#include "ext.h"            // standard Max include
#include "ext_obex.h"       // required for "new" style objects
#include "z_dsp.h"          // required for MSP objects

class MSPExt;               // the C++ style class

extern "C" {            // struct to represent the object's state
  typedef struct _simple {
    t_pxobject    ob; // the object itself (t_pxobject in MSP)
    MSPExt*    me;
    void* out_sig;    // outlets
  } t_simple;

  // method prototypes
  void *simple_new(t_symbol *s, long argc, t_atom *argv);
  void simple_free(t_simple *x);

  void simple_assist(t_simple *x, void *b, long m, long a, char *s);
  void simple_float(t_simple *x, double f);

  // for MSP
  void simple_dsp(t_simple *x, t_signal **sp, short *count);
```

[21] Max also interfaces with Java and JavaScript, and externals exist to extend it to Lua (`http://www.mat.ucsb.edu/~wakefield/lua~/lua~.htm`), Ruby (`http://compusition.com/software/maxmsp/jruby_for_max`) etc. We use the C API for sample-accurate, online algorithms, since this is the closest option to the native Max/MSP objects, which are also C-based.

[22] see `https://cycling74.com/downloads/sdk` .

```
  void simple_dsp64(t_simple *x, t_object *dsp64, short *count, double
    samplerate, long maxvectorsize, long flags);
  t_int *simple_perform(t_int *w);
  void simple_perform64(t_simple *x, t_object *dsp64, double **ins, long numins,
    double **outs, long numouts, long sampleframes, long flags, void *userparam);

  static t_class *simple_class = NULL;  // global class pointer variable
} // end extern "C"

// C++ style class starts here:
class MSPExt {
public:
  t_simple *max;

  // internal vars:
  float offset;

  MSPExt();
  ~MSPExt();
  void perform(double *in, long sampleframes, int dest);

  // inlet methods:
  void float(double f);
};
```

LISTING 2.1: simple~.h, the C++ MSP external template header.

```
#include <simple~.h>

extern "C"
{
  int C74_EXPORT main(void) {
    t_class *c = class_new("simple~", (method)simple_new, (method)simple_free,
    (long)sizeof(t_simple),
      0L, A_GIMME, 0);

    class_addmethod(c, (method)simple_float, "float", A_FLOAT, 0);
    class_addmethod(c, (method)simple_dsp, "dsp", A_CANT, 0);  // Old
    32-bit MSP dsp chain compilation for Max 5 and earlier
    class_addmethod(c, (method)simple_dsp64, "dsp64", A_CANT, 0);  // New
    64-bit MSP dsp chain compilation for Max 6
    class_addmethod(c, (method)simple_assist, "assist", A_CANT, 0);

    class_dspinit(c);
    class_register(CLASS_BOX, c);
    simple_class = c;
```

```
  return 0;
}

void *simple_new(t_symbol *s, long argc, t_atom *argv) {
  t_simple *x = (t_simple *)object_alloc(simple_class);
  x->me = new MSPExt();
  if (x) {
    x->me->max = x;
    dsp_setup((t_pxobject *)x, 1);    // MSP inlets: arg is # of inlets
    // outlets, RIGHT to LEFT:
    x->out_sig = outlet_new(x, "signal");     // signal outlet (note "signal"
  rather than NULL)
  }
  return (x);
}

void simple_free(t_simple *x) {
  dsp_free((t_pxobject*)x);
}

void simple_assist(t_simple *x, void *b, long m, long a, char *s) {
  if (m == ASSIST_INLET) { //inlet
    sprintf(s, "I am inlet %ld", a);
  }
  else {  // outlet
    sprintf(s, "I am outlet %ld", a);
  }
}

void simple_float(t_simple *x, double f) {
  x->me->float(f);
}

void simple_dsp(t_simple *x, t_signal **sp, short *count) {
  dsp_add(simple_perform, 4, x, sp[0]->s_vec, sp[1]->s_vec, sp[0]->s_n);
}

void simple_dsp64(t_simple *x, t_object *dsp64, short *count, double
  samplerate, long maxvectorsize, long flags) {
  object_method(dsp64, gensym("dsp_add64"), x, simple_perform64, 0, NULL);
}

t_int *simple_perform(t_int *w) { // 32-bit perform method for Max 5
  t_simple *x = (t_simple *)(w[1]);
  t_float *in = (t_float *)(w[2]);
  t_float *out = (t_float *)(w[3]);
  int n = (int)w[4];
  x->me->perform((double *)in, (double *)out, n);
```

```
    return w + 5;
  }

  void simple_perform64(t_simple *x, t_object *dsp64, double **ins, long numins,
    double **outs, long numouts, long sampleframes, long flags, void *userparam)
    {
    t_double *in = ins[0];     // audio inlet #1
    t_double *out = outs[0];   // audio outlet #1
    int n = sampleframes;
    x->me->perform(in, out, n);
  }
} // end extern C

// now we can implement our functionality in C++ style:
MSPExt::MSPExt() { offset = 0.0; }
MSPExt::~MSPExt() {}

void MSPExt::perform(double *in, double*out, long sampleframes) {
  while (sampleframes--)
    *out++ = *in++ + offset;
}

void MSPExt::float(double f) { offset = f; }
```

LISTING 2.2: simple~.cpp, the C++ MSP external template source.

2.5.3 Performance Tracking Systems

We review several performance tracking systems in use in the current decade. They range from Max objects aimed at rock and pop music (*Performance Follower* [201]) or new music (*Antescofo* [49]) interaction, through symphonic orchestra followers (PHENICX [10]), to commercial practice and learning packages (Cadenza[23], Yousician[24], Tonara[25]). We position them alongside our proposed rvdtw~ system in a summary comparison in table 2.1.

Antescofo

Antescofo is an award-winning software[26] enabling the authorship (programming) and performance (execution) of mixed music, developed within the Music Representations

[23] see `http://www.sonacadenza.com`.
[24] see `http://get.yousician.com`.
[25] see `http://tonara.com/tonara`.
[26] see `http://repmus.ircam.fr/antescofo`.

	rvdtw~	*Antescofo*	*Performance Follower*	Arzt et al. 2015 (PHENICX)	Cadenza (ex-*Music Plus One*)	Yousician	Tonara
Platform	Max	Max, Pd	Max	OSX+iOS	iOS	Win, OSX, Linux, iOS, Android	iOS
Symbolic reference	no	yes	no	yes	yes	yes	yes
Audio reference	yes	no	no	yes (multiple)	yes	no	no
Accepted inputs	any	any	any*	any	9 instr.**	4 instr.**	any***
IMS support	yes	yes	yes	no	no	no	no
Auto accomp.	yes	yes	yes	no	yes	yes	no
Reactive language	no	yes	no	no	no	no	no
Integrated notation	no	yes	no	yes (PHENICX)	yes	yes	yes
Practice software	no	yes****	no	no	yes	yes	yes

TABLE 2.1: Comparison of performance tracking / interactive music systems.
* suited for beat-centric material such as rock and pop music.
** unclear whether different models for instruments, or the scope of the score library.
*** score library covers 4 classical instruments.
**** includes tutorials for mixed music design and performance.

team at IRCAM. Its listening machine consists of a predictive coupled position/tempo two-agent architecture, informing each other in real time [50]. Since the program evaluation is a product of human-machine dynamics, *Antescofo* enables the authoring of dynamic mixed music scenarios in the sense described in section 2.1: cyber-physical systems where human musicians are integrated in the computing loop [55].

This powerful score-driven IMS platform has been widely adopted by artists and used in numerous new music creations and performances worldwide.[27] Its dedicated visual interface and notation framework is *AscoGraph* [46].

The *Antescofo* software features extensively in this thesis. We review the basic system architecture and composition workflow in section 2.5.4. Our contributions to the research and development of *AscoGraph* form the substance of chapter 3. Finally, we present new work powered by the system's synchronisation and coordination engine in section 4.2.

[27] An incomplete list is available at `http://repmus.ircam.fr/antescofo/repertoire` .

Starting in 2016, Antescofo is a start-up company working towards a multi-platform standalone app[28].

Performance Follower

This system was developed by Adam Stark as part of his PhD research [199]. Powered by a beat-synchronous chord analysis framework [202] and a performance-driven prediction engine [201], the *Performance Follower* is able to control virtual instruments that adequately accompany a musician without access to any reference, symbolic or sonic. Indeed, this is the only instance in this review of systems that do not perform any score following or audio alignment, and instead act solely on performance analysis.

Stark's follower was designed for and evaluated with rock and pop styles of instrumental input, mainly direct electric guitar [199]. Its ability to leverage beat and harmonic data to drive an accompaniment inspired us to incorporate a beat tracking module into our own tracker, as we describe in section 4.1.5.

PHENICX

The PHENICX (Performances as Highly Enriched aNd Interactive Concert eXperiences) project hosted by UPF Barcelona[29] is a multi-modal framework aimed at enriching the classical concert experience by extracting and displaying various types or information to the audience in real time. One of the project's main activities was to leverage Arzt and Widmer's multi-agent online DTW-based follower [7] to track an orchestra during a performance of Richard Strauss' *Alpensinfonie* in order to continuously display the current position on the sheet music.

The software is currently unavailable to the public, and as such cannot be applied to different tasks such as automatic accompaniment or interactive music.

Practice and Tuition Software

Cadenza, Yousician and Tonara (as well as other similar apps like Rocksmith and SmartMusic) are also closed software, designed by their manufacturers for specific learning and rehearsal tasks. The only available information about the inner workings of Yousician and Tonara comes from patents [112, 119].

[28] see `http://www.antescofo.com` .
[29] see `http://phenicx.upf.edu` .

Meanwhile, we do have more insight into Cadenza's listening machine due to Christopher Raphael's research for its academic precursor, *Music Plus One* [171]. Synchronisation is achieved through HMMs, with a Kalman filter-like predictive model informing the real-time matching. The model is calibrated with each subsequent rehearsal, learning the musician's particular tendencies over each specific piece. In this sense, *Music Plus One* acts as an evolving accompanist.

2.5.4 Dynamic Mixed Music Composition in *Antescofo*

We saw in figure 2.4 that, in a realisation of *Anthemes II*, the temporal ordering of the audio processes is implicit in the stepwise evaluation of the score's data-flow graph, based on the human operator's manual triggering of cues. But the audio processes' activation, their control and most importantly their interaction with respect to the physical world (the human violinist) are neither specified nor implemented.

The authoring of time and interaction of this type, and its handling and safety in real-time execution is the goal of *Antescofo* system and language [49, 78], which couples a listening machine [50] and a reactive engine [79] in order to dynamically perform the computer music part of a mixed score in time with live musicians. This highly expressive system is built with time-safety in mind, supporting musical-specific cases such as musician error handling and multiple tempi [54, 80]. Actions can be triggered synchronously to an event $e(t)$ detected by the listening machine (these are called *atomic actions*), or scheduled relative to the detected musician's tempo or estimated speed $\dot{e}(t)$. Finally, a real-time environment (Max/MSP, Pd or another OSC-enabled responsive program) receives the action commands and produces the desired output. The *Antescofo* runtime system's coordination of computing actions with real-time information obtained from physical events is outlined in Figure 2.10.

Antescofo's timed reactive language [51] specifies both the expected events from the physical environment, such as polyphonic parts of human musicians (as a series of EVENT statements) and the computer processes that accompany them (as a series of ACTION statements), to form *augmented scores*. Our brief review touches on several aspects of the language; for a detailed specification please consult the *Antescofo* reference manual [91]. The syntax is further described in [52], while a formal definition of the language is given in [80].

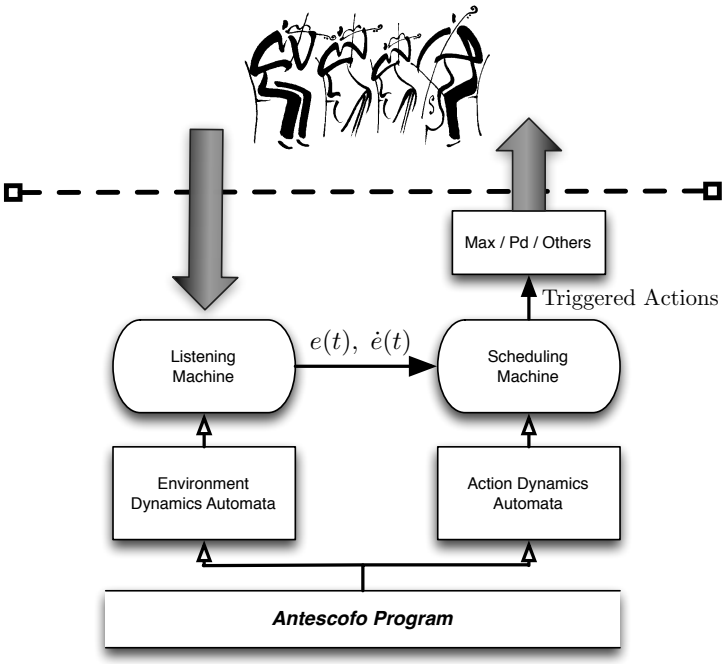


FIGURE 2.10: *Antescofo* execution diagram (reproduced from [52]).

To facilitate the authoring and performance of *Antescofo* scores, a dynamic visualisation system was conceived, with a view to realising a consistent workflow for the *compositional* and *execution* phases of mixed music. *AscoGraph* [46] is the dedicated user interface that aims to bridge the three notational paradigms delineated in section 2.2: *symbolic*, *integrated* and *dynamic*. We next demonstrate the first two aspects with a brief study of *AscoGraph*'s basic notation model, before laying out different strategies to tackle the dynamic dimension in the next chapter.

2.5.5 The Basic *AscoGraph* Visualisation Model

Since its inception, *AscoGraph*'s visual model has been centred around the *actions view* window, which is aligned to the instrumental *piano roll* by means of a common timeline. Electronic actions are either discrete (visualised by circles) or continuous (curves) but they are all strongly timed[30] [50]. Figure 2.11 displays the implementation of *Anthèmes II* (Section 1) from the *Antescofo Composer Tutorial*[31]. This layout facilitates the authoring process by lining up all the elements according to *musical time*, which is

[30]Wang [216] defines a *strongly timed* language as one in which there is a well-defined separation of synchronous logical time from real-time. Similarly to Wang's *ChucK* language, *Antescofo* also explicitly incorporates time as a fundamental element, allowing for the precise specification of synchronisation and anticipation of events and actions.

[31]available at `http://forumnet.ircam.fr/products/antescofo/`.

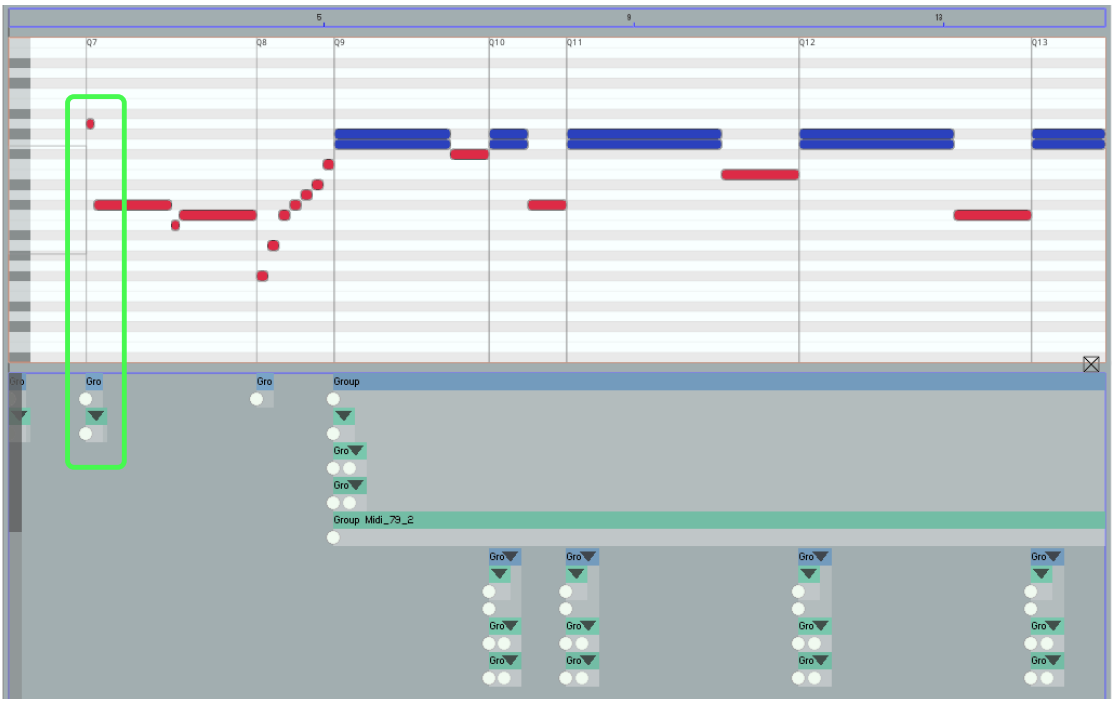


FIGURE 2.11: *AscoGraph* visualisation for *Anthèmes II* (Section 1) from the *Antescofo Composer Tutorial*. Top: piano roll; Bottom: action view. The left-hand portion highlighted by a rectangle corresponds to Listing 2.3.

independent of the *physical (clock) time* of performance. Thus, in cases where the score specifies a delay in terms of seconds, this is automatically converted to bars and beats (according to the local scored tempo) for visualisation.

Generally, atomic actions in the *Antescofo* language have the following syntax: [<delay>] <receiver_name> <value>. An absent <delay> element is equivalent to zero delay, and the action is assumed to share the same logical instant as the preceding line.

```
NOTE 0 1.0   ; silence
NOTE 8100 0.1    Q7
  Annotation "Harms open"
  hr-out-db -12.0 25   ; bring level up to -12db in 25ms
  group harms { ; four harm. pitches:
    hr1-p (@pow(2., $h1/12.0))
    hr2-p (@pow(2., $h2/12.0))
    hr3-p (@pow(2., $h3/12.0))
    hr4-p (@pow(2., $h4/12.0))
  }
```

LISTING 2.3: Opening notes and actions in the *Anthèmes II* augmented score.

Code listing 2.3 shows the starting note and action instructions of the score. After one beat of silence, the first violin note triggers the opening of the harmonizer process, by way of a nested group of commands. The resulting hierarchy is reflected in the green-highlighted section of Figure 2.11's action view: the first action group block on the left contains a white circle (representing two simultaneous messages) and a sub-group block, which in turn includes a circle (containing four messages to the harmonizer units). Note the absence of any time delay: all atomic actions mentioned above are launched in the same logical instant as the note detection.

This visualisation model meets the first notational goal we specified at the end of section 2.2: it acts as a graphic map of the piece, which reveals itself through interaction (e.g. hovering the mouse over message circles lists their contents). The remaining goals of composition, testing and performance, motivate our pursuit to represent dynamic processes in chapter 3 below.

2.6 Summary

In this chapter we have outlined research relevant to this thesis. We surveyed the fields of interactive computer music performance, notation and design, of real-time synchronisation and of data sonification, and the related technology. Particularly, we focused on the online DTW algorithm and the *Antescofo* IMS platform, upon which a large portion of our work relies.

Chapter 3

Information Visualisation of Dynamic Sound Processes

The aim of information visualisation is to organise heterogeneous data graphically in a way that optimises human cognition, based on an inherent model [34]. The model should address specific visualisation aims and yet be general enough to allow for wide public adoption and adaption to other problems from the domain. An interesting instance is the "standard" model of Western music notation: it developed over centuries to render pitch and rhythm readable, writable and playable by its users, while its semantics have been further extended to wider harmonic, instrumental and stylistic applications over the years, all the time preserving a basic framework. Still, symbolic notation will never capture all the sonic aspects a musician might have in mind when authoring or reading a score. In this sense, any visualisation must strike a bargain between what is shown the user, and what remains hidden or implied. The major advantage of interactive models is the enabling of user-led exploration and focus, so that the visible domain becomes fluid, adapting to one's own needs and preferences.

In this chapter we first address the problem of ergonomically representing the timed hierarchical structures present in *AscoGraph*, showing how they relate to more general visualisation principles and techniques.

We then extend this design study to include the visualisation of scored dynamic live electronic processes. The visual framework presented in this chapter is the outcome of a continuous workshopping cycle involving the mixed music composer community in

and around IRCAM. While many features are active in the current public version of *AscoGraph*[1], others are in the design, development or testing phases. This is apparent in several proof of concept images, which are a blend of the publicly available *AscoGraph* visualisation and mockups of the features under construction. We can expect the final product to contain minimal differences from the present presentation.

3.1 Hierarchical Code Structures in *AscoGraph*'s Action View

The following is a list of design requirements, to be tackled in subsequent sections. They are ordered by visual scale, from the level of action groups to the score as a whole:

- clearly represent action group content and duration. We frame this issue as a *strip packing* problem in section 3.1.1, and we propose three new algorithms to solve it in section 3.1.2;
- ensure time coherence between atomic and compound action items. Section 3.1.3 covers this requirement;
- facilitate the locating of specific information. We achieve this through action filtering, presented in section 3.1.4;
- represent the hierarchical structure of the code. Sections 3.1.3 and 3.1.5 show our advances in this regard;
- synthesise a useful overview of the score, showing the degree of electronic complexity over musical time. All the sections listed above contribute to this aspect, which is further discussed in the concluding section.

Later on in this chapter (section 3.4) we approach the hierarchical dimension directly by proposing an out-of-time model which maps out the code structure while hiding the temporal dimension.

3.1.1 The Bin Packing Perspective

In the previous iteration of *AscoGraph* [46], it was common to have temporally overlapping action groups displayed on top of each other along the timeline. The resulted

[1] at the time of writing, the newest release of *Antescofo* is v0.9, which includes *AscoGraph* v0.2.

in a loss of coherence and clarity, with the widths of some action blocks no longer corresponding to their programmed durations.

In order to rectify this, we amended the model by stacking action groups in downward non-overlapping order, similarly to how the elements *within* groups are arranged. In the design process, we faced two challenges: (1) managing the 2D space efficiently (due to the new tendency towards vertical growth) and (2) maximising readability and easy navigation. The solutions we propose all strike a certain balance between these parameters.

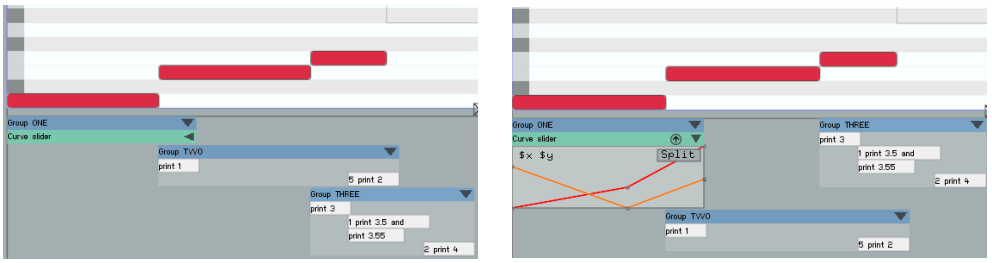
For arranging rectangular blocks in a two-dimensional space, we translate the issue to a *strip packing* problem. A subset of the *bin packing* approach, strip packing is used in areas ranging from optimizing cloth fabric usage to multiprocessor scheduling [206]. Algorithms seek to arrange a set of rectangles within a 2D space of fixed width and bottom base, and of infinite height. In our present case, the width of the strip corresponds to the total duration of the piece, and the rectangles to be placed are the action group blocks.

Bin packing is a rapidly expanding research area, with results often being difficult to track and compare [172]. A good measure of algorithm efficiency is the *absolute approximation ratio* $\sup_I \mathrm{ALG}(I)/\mathrm{OPT}(I)$, where $\mathrm{ALG}(I)$ is the strip height produced by the algorithm ALG on an input I, and OPT is the optimal algorithm. By calculating a superior bound for any input we obtain a useful value for any input size [97].

Unlike existing bin packing problems, all *AscoGraph* action blocks must retain their X coordinate along the time axis. This constraint alone distinguishes our problem from the rest of the bin packing literature. Even multiprocessor scheduling strategies involve expediting or delaying tasks in time [206]. Thus, relying on existing algorithms becomes impractical.

3.1.2 Packing Algorithms

We introduce three new algorithms for stacking action groups in *AscoGraph*'s graphical editor. The user can switch between each and the original display style through the application's *View* menu. The appropriate option will depend on score complexity and the user's personal taste.



(A) Curve block is closed (B) Curve block is open

FIGURE 3.1: Expanding an action block leaves room for another block to drop to the horizontal base.

The interactive nature of the software demands dynamic and periodic calls to block arrangement calculation and rendering. For instance, expanding an action group to reveal its contents might free up space next to it – see figure 3.1. In order to minimise user confusion and computer processing overhead, we take several measures. Firstly, the ordering is computed for all actions in the score, and not just the ones in a specific timeframe; that way, when scrolling through a score at a constant zoom level, recalculation is not necessary. Secondly, we defined a number $n = 10$ of user actions between recalculations. Finally, the most recently clicked action group always stays in place, no matter what reordering process is triggered otherwise.

It is important to note that following bin packing conventions, we shall consider boxes as being placed *on top of* the strip base. Naturally, in the *AscoGraph* environment the situation is mirrored and we build *downwards* starting from the upper border.

In this section, we define a set of rectangles $I = \{r_1...r_n\}$, with each r_i being determined by height h_i, width w_i and start time x_i.

First Fit (FF)

The first option is the trivial solution of placing the blocks in the first space they will fit, starting from the base. The main strengths of this algorithm are speed and predictability: blocks are placed in the order in which they appear in the source code text, which is also their scheduled temporal order. Since spatial position is in principle the strongest perceptual cue[127], if all groups are considered equally important then it makes sense to place the first occurring elements in the top positions. In section 3.1.4 we propose filtering as a way to express precedence of certain blocks over others. Other options in this direction remain to be examined.

(A) FF (B) OPT (FFD)

(C) FFD (D) OPT (FFDT)

FIGURE 3.2: Horizontally constrained strip packing. The rectangle set in (a) and (b) is first arranged by First Fit, and then optimised by FFD. The set in (c) and (d) is first stacked by FFD, then optimised by FFDT.

The downside is evident in figure 3.2a and 3.2b. Let us consider the worst-case scenario in order to calculate the absolute approximation ratio. Let $I' = \{r'_1 ... r'_n\}$ with increasing heights $h'_i = h'_{i-1} + \varepsilon_i$ and for simplicity, let all rectangle widths be equal. While FF would stack them on top of each other (see figure 3.2a), the optimal configuration would stack them two by two (figure 3.2b), so that the maximum height is given by the final two elements:

$$\frac{\mathrm{FF}(I')}{\mathrm{OPT}(I')} = \frac{\sum_{i=1}^{n} h'_i}{h'_{n-1} + h'_n} \tag{3.1}$$

It is obvious that a finite upper bound cannot be defined.

First Fit Decreasing (FFD)

Note that in the previous case, the optimal configuration could be reached by simply reordering the blocks by height. This leaves room for blocks of smaller height to drop to the baseline, as we saw in figures 3.1 and 3.3b.

This insight lies at the root of the classic FFDH strip packing algorithm [45]. In our case, the FFD algorithm orders the blocks by non-increasing height, after which the First Fit process is applied.[2] Figure 3.2c shows a basic example of an FFD arrangement, along with its optimal solution at figure 3.2d.

[2]The difference to the classic FFDH algorithm is the absence of horizontal levels. New blocks are stacked at the minimum possible altitude rather than a common level.

It is obvious that, for any $n = 4$ blocks, the above configuration is the one that produces the largest FFD(I)/OPT(I) ratio, by means of the gap between blocks 3 and 4. Generalizing, we can state the following:

Definition 1. Given an FFD layout, a pair of non-adjacent blocks is called a *basic broken pillar*, if the horizontal projection of one block, traversing upwards, intersects the other block.

In figure 3.2c, blocks 2 and 3 form a basic broken pillar.

Definition 2. In an FFD layout, a *broken pillar* is a chain of basic broken pillars where the top block of one is the bottom block of the next.

Theorem 1. *For any configuration of blocks I,*

$$FFD(I) - OPT(I) \leq g, \tag{3.2}$$

where g is the accumulated vertical gap inside the tallest broken pillar.

Proof. Since, for a basic broken pillar, the reduction to be made by layout optimisation is evidently limited by the basic pillar's internal gap, we can proceed along the chain to obtain the overall maximum reduction. □

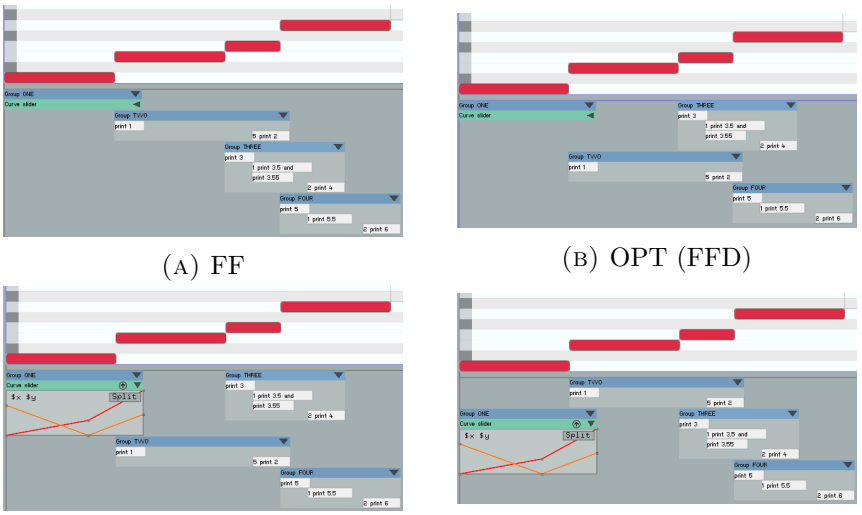
Corollary 1. *For any $n > 4$, the configuration that maximises $FFD(I)/OPT(I)$ is generated by sequentially adding to a broken pillar exclusively.*

Proof. When placing each block, if it is not adding to a broken pillar, then the difference between FFD(I) and OPT(I) remains constant. The goal must be to maximise the accumulated gap for the given set of blocks. □

Corollary 2. *In a broken pillar, if the accumulated vertical gap g is greater than the accumulated height of the member blocks, then g will not contribute to layout optimisation.*

Proof. For a basic broken pillar that satisfies these conditions[3] the conclusion is obvious. Proceeding along the chain, each gap's contribution will continue to be annulled by the stack on the opposite side. □

[3]The minimum number of blocks to build a basic broken pillar is 5. 2 blocks form the broken pillar itself, and at least 3 blocks are needed (under FFD conditions) to build a large enough gap.



(A) FF (B) OPT (FFD)

(C) FFD (D) OPT (FFDT)

FIGURE 3.3: The three *AscoGraph* packing options: (a) FF - action group blocks are placed as close to the horizontal timeline as possible, in the order in which they appear in the code. (b, c) FFD - blocks are ordered by height before being placed as in FF. (d) FFDT - blocks are ordered according to a gap-minimisation heuristic before being placed on the timeline as in FFD.

It is now possible to present an upper bound to the absolute approximation ratio.

Theorem 2. *The absolute approximation ratio of the FFD algorithm is lower than 2.*

Proof.

$$\sup_I \frac{\mathrm{FFD}(I)}{\mathrm{OPT}(I)} \leq \frac{h_{pillar} + g}{h_{pillar}} = 1 + \frac{g}{h_{pillar}} \tag{3.3}$$

where g is the accumulated gap inside the tallest broken pillar. If $g \geq h_{pillar}$, then its contribution is annulled and we must look to the second tallest broken pillar. And so on, until the fraction becomes subunitary. □

First Fit Decreasing Towers (FFDT)

As before, the optimal configuration in the previous example hints at the following algorithm. We propose a greedy heuristic that builds upon FFD while tackling situations common in *AscoGraph*, such as one action block sharing time with several blocks on both sides of it. (e.g. figure 3.2c and d). The basic goal is to minimise the gaps in broken pillars.

The FFDT algorithm first orders all blocks as in FFD. Then, action group *towers* are defined at the time-axis intersections between two or more group blocks. Their height

is equal to the sum of the heights of their component blocks. For instance, in figure 3.2c and d the rectangle 2 is part of four towers: $\mathsf{T}\{r_1, r_2\}, \mathsf{T}\{r_2\}$[4], $\mathsf{T}\{r_2, r_3\}$ and $\mathsf{T}\{r_2, r_3, r_4\}$.

The entire width being now split along these virtual vertical strips we call towers, we are now able to refine the ordering of the blocks. Algorithm 1 below lists all the steps.

Algorithm 1 FFDT

(1) Compute all tower assignments and heights.
(2) For each block r_i, compute:
- MTH_i = the maximum tower height among the towers containing the block;
- NT_i = the number of towers containing the block;

(3) Order all action group blocks as follows:
(3a) by decreasing MTH_i
if equal **then**
  (3b) by decreasing NT_i;
  if equal **then**
    (3c) by non-increasing block height (as in FFD).
  end if
end if
(4) Place blocks using FF.

The maximum tower height is a definite lower bound of any *AscoGraph* strip packing configuration. Therefore, in the FFDT heuristic the top tower will always be placed first, in an attempt not to overshoot this lower bound. Among its component blocks, the ones that are shared with many other towers are then given preference – the intention again being to maintain tower integrity as much as possible. Lastly, as with FFD, tall blocks are prioritised so as to fill gaps efficiently.

We assert that the FFDT algorithm's absolute approximation ratio is lower than 2 (similarly to FFD), and it is safe to assume it is much closer to 1.

3.1.3 Time Coherence of Atomic Actions

As we mentioned in section 2.5.5, atomic actions are instantaneous—which means that ideally they should not take up horizontal space. Moreover, as seen in figures 3.1 and 3.3, they introduce unnecessary clutter in the workspace.

Our solution is to group all instances from a specific hierarchical level and display them on a single line as small circles, or conceptual *points*. When the mouse hovers over such

[4]The minimal tower is one that is identical to an (isolated) action block.

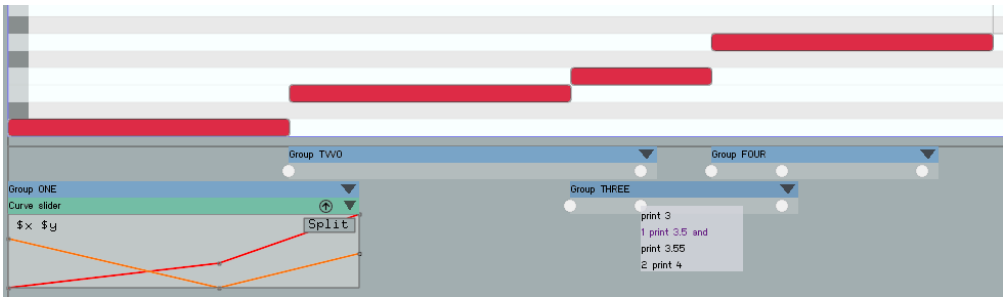


FIGURE 3.4: Time-coherent message circles display

a point, a list of the messages it contains is shown. Figure 3.4 shows the expanded list for Group THREE; the 4 messages are spread over 3 different points in time, which is why 3 circle are present in the message line.

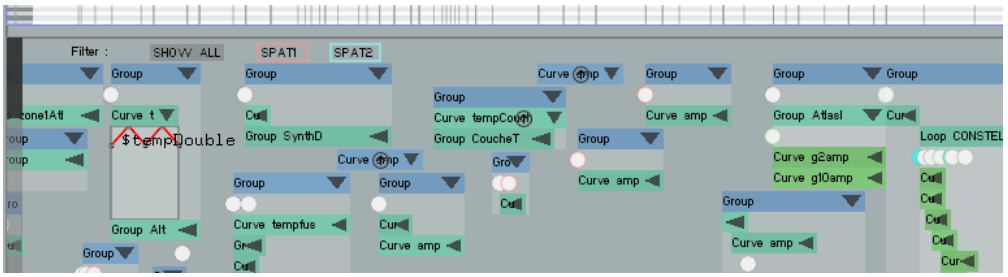
This feature also harnesses the visualisation principle of explicit linking [181]. When the user clicks on a certain message in the expanded list, the corresponding line of code is highlighted. Thus, we capitalise on the coupling between adjoining views and their value is enhanced. This feature was already present in *AscoGraph* among other elements: action blocks, musical events in the piano roll, and specific lines of code are all linked and clickable.

This new atomic action model is fully time coherent and considerably clearer than before. The user experience improvement over the classic model becomes most obvious when dealing with complex scores with many messages.

3.1.4 Track Filtering

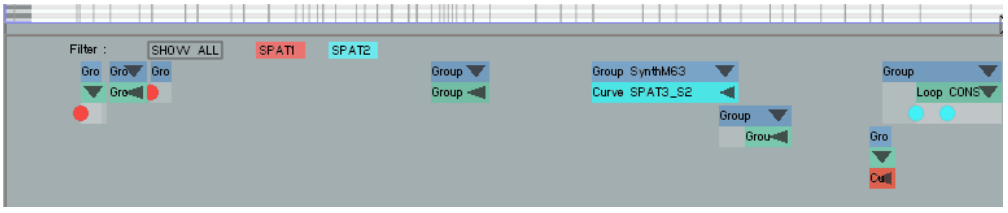
AscoGraph implements the "visual information seeking mantra" – overview first, zoom and filter, details on demand [195]. While zooming the timeline and expanding blocks to reveal their contents have already been mentioned, filtering is another function essential to information seeking and score navigation.

In *Antescofo*, filters are defined in the score code through regular expressions. The resulting set of objects that match a certain user-defined expression is referred to as a *track* [91]. The user can define up to 32 such tracks. Figure 3.5 illustrates the use of filters. Note that any number of tracks can combine into a filter. At any point, the user can click the SHOW ALL button to return to the initial view.



(A) No filtering: all group blocks are shown.

(B) One filter: SPAT1 track is active.



(C) Two filters: SPAT1 and SPAT2 tracks are active.

FIGURE 3.5: Track filtering in *AscoGraph*: A line is added just below the timeline showing all available track names which the user can filter by.

We return to the subject of filters and tracks in section 3.3.2, where we propose an updated *simulation view* that graphically traces tracks' component states over time using staff-based notation.

3.1.5 Action Block Colours

In designing the colour schemes for *AscoGraph*, we observed the guidelines of Tufte [212] that large areas should use desaturated colour and that high contrast between foreground and background improves readability. Here we note an additional contribution over the previous version of *AscoGraph*, in which each action block header was assigned a random colour. In our updated model, header colours signify (1) hierarchy and (2) tracks. Unless a block header's colour is overridden by a filter highlight, it will correspond to the block's hierarchical level. First-order action groups have blue headers; afterwards we vary the hue angle in 45 degree increments. Filter highlight tones are purer and more saturated in order to clearly distinguish them from default hierarchical colours. Message circles

are by default transparent white, regardless of hierarchy. Once allocated into tracks, their colours also change accordingly.

We now turn to our study of dynamics in the *Antescofo* reactive language, and their visual representation within *AscoGraph*.

3.2 Dynamic Elements in the *Antescofo* Language

As we established in section 2.1, dynamic behaviour in computer music can be the result of both acoustic input during live performance, and algorithmic and dynamic compositional elements prescribed in the score itself. Accordingly, in an *Antescofo* program, dynamic behaviour is both produced by real-time (temporal) flexibility as a result of performing with a score follower, and through explicit reactive constructs of the action language.

In the former case, even though the temporal elements can be all statically defined in the score, they become dynamic during live performance due to the tempo fluctuations estimated by the listening machine. We touched on this basic dynamic aspect in our discussion of machine listening within IMS from section 2.1, as well as in our introduction to *AscoGraph* from section 2.5.5, where we noted the implicit representation of physical time as musical time.

The second case employs the expressive capabilities of the strongly timed action language of *Antescofo*. Such explicitly dynamic constructs form the topic of this section.

3.2.1 Run-time Values

Variables in the *Antescofo* language can be run-time, meaning that their values are only determined during live performance (or a simulation thereof). The evaluation of a run-time variable can quantify anything as decided by the composer, from a discrete atomic action to breakpoints in a continuous curve, as shown in code listing 3.1.

In this basic sample, the value of the `level` output can be the result of an expression defined somewhere else in the code, whose members depend on the real-time environment.

In the example on the right, the output level is defined by $y, which grows linearly over 2 beats from zero to the run-time computed value of $x.

```
NOTE C4 1
level $x
```

```
NOTE D4 1
curve level {
  $y {
        ( 0 )
    2   ( $x )
  }
}
```

LISTING 3.1: Dynamic amounts.
Left: atomic value. Right: curve breakpoint.

Thus, while for now the circle-shaped action message display from section 2.5.5 is adequate for the dynamic atomic action, for the curve display we need to introduce a visual device that explicitly shows the target level as being dynamic. Our solution is described in section 3.3.1. Additionally we propose an alternative treatment of both atomic actions and curves, in the context of performance traces, in section 3.3.2.

3.2.2 Durations

On *AscoGraph*'s linear timeline, we distinguish two kinds of dynamic durations. Firstly the *delay* between two timed items, such as an EVENT and its corresponding ACTION, or between different breakpoints in a curve; and secondly, the number of *iterations* of a certain timed block of instructions.

The examples in code listing 3.2 show the two kinds of temporal dynamics: On the left, the run-time value of $x determines the delay interval (in beats, starting from the detected onset of NOTE C4) until level receives the value 0.7, and the duration of the (continuous) increase from 0 to 0.5. On the right, the duration of execution of the loop and forall structures depends, respectively, on the state of $x and the number of elements in $tab. In these cases the terminal conditions of loop and forall are reevaluated on-demand.

```
NOTE C4 1
$x level 0.7
NOTE D4 1
curve level {
  $y {
    ( 0 )
    $x  ( 0.5 )
  }
}
```

```
NOTE E4 1
loop L 1 {
$x:=$x+1
} until ($x > 3)

NOTE F4 1
forall $item in $tab {
$item level ($item * 2)
}
```

LISTING 3.2: Dynamic durations.
Left: delay durations. Right: number of iterations.

A particular extension of iterative constructs are *recursive processes*. A process is declared using the @proc_def command, and can contain calls to itself or other processes. Thus, the behaviour and activation interval (*lifespan*) of a process, once it has been called, can be highly dynamic. The example in code listing 3.3 produces the same result as the loop block in code listing 3.2. See [91] for in-depth specifications of all iterative constructs.

```
@proc_def ::L() {
  if ($x <= 3) {
    1 ::L() ; new proc call
  }
  $x:=$x+1
}
NOTE E4 1
::L() ; initial proc call
```

LISTING 3.3: Dynamic recursive process.

We introduce a graphic solution for representing dynamic durations in section 3.3.1. Going further, the action view model is not well suited for dynamically iteration-based repetition. We propose three alternatives: "unpacking" such constructs into an execution trace (section 3.3.2), detailing their structure in a tree-based graph (section 3.4.1), and monitoring their status in an out-of-time auxiliary panel (section 3.4.2).

3.2.3 Occurence

The examples we have shown thus far, while pushing the limits of *AscoGraph*'s action view, can still be represented along a linear compositional timeline. There is a third category which could not be drawn alongside them without causing a breakdown in temporal coherence, as shown in code listing 3.4.

```
whenever ($x) {
  level 0
} during [2#]
```

Listing 3.4: Dynamic occurence point.

Here, the action block is fired whenever the variable $x is updated. Moreover, this is set to occur only twice in the whole performance, hence the during [2#]. From here, it is easy to imagine more complications – recursive process calls, dynamic stop conditions etc – leading to a highly unpredictable runtime realisation.

In most cases, since such occurence points are variable, nothing but the overall lifespan of the whenever (from entry point down to stop condition fulfilment) should be available for coherent representation; this might still be helpful for the composer, as we show in section 3.3.1.

Since whenever constructs are *out-of-time* dynamic processes[5], being detached from the standard timeline grid, they require new methods of representation beyond the classic action view. We discuss the "unpacking" of whenever blocks onto traces in section 3.3.2, and their non-timeline based representations in section 3.4.

3.2.4 Synchronisation Strategies

In *Antescofo*, the tempo of each electronic action block can be dynamically computed relatively to the global tempo detected by the listening machine. Through attributes

[5]Our use of the term "out-of-time" is derived from the sense coined by Xenakis [222] to designate composed *structures* (and the methods used to generate them), as opposed to sonic form. In an analogous fashion, we distinguish "in-time" electronic actions that are linked to specific acoustic events from "out-of-time" constructs, which are not. Just like Xenakis' structures, during the performance, the out-of-time constructs are actuated (given sonic form) in time. The difference from Xenakis' approach is that *Antescofo* out-of-time actions do not reside on a separate plane of abstraction from in-time actions: only the nature of their activation is different.

attached to an action group, its synchronisation can be defined as @loose, @tight, or tied to a specific event @target; the latter enabling timing designs such as real-time tempo canons [211]. Since @target-based relationships define temporal alignment between a group and event(s), we can visualise this by connecting the piano roll to the action tracks (see section 3.3.2), or by drawing the connections in an out-of-time model (see section 3.4.1).

Additionally, the *Antescofo* language allows for dynamic targets, acting as a moving synchronisation horizon. In this case, the tempo is aligned to the anticipation of the @target event at a specific distance in the future, computed either by a number of beats or a number of events. We can indicate this synchronisation lookahead in relation to the timeline; as shown in the case study in section 3.5.2 for an example.

3.2.5 Score Jumps

The instrumental events in an *Antescofo* score are implicitly a sequence of linear reference points, but they can be further extended to accommodate jumps using the @jump attribute on an event. Jumps were initially introduced to allow simple patterns in western classical music such as free repetitions, or *da capo* repetitive patterns. However, they were soon extended to accommodate composers willing to create open form scores [88].

For the purposes of visualisation, we distinguish two types of open form scores: in the first, jump points are fixed in the score (static), while their activation is left to the discretion of the performer. This scheme is more or less like the *da capo* example

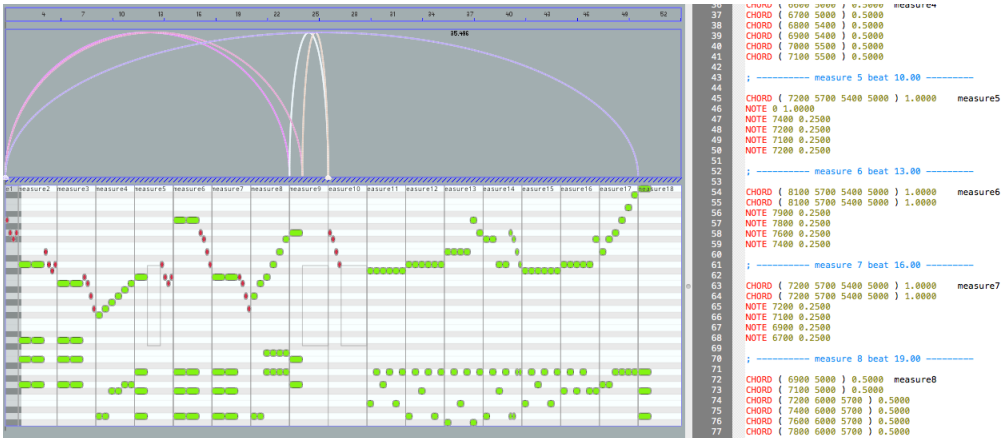


FIGURE 3.6: *Ascograph* with static jumps: A classical music score (Haydn's *Military Minuet*) with *Antescofo* jumps simulating *da capo* repetitions during live performance.

in figure 3.6. Its success in live performance depends highly on the performance of the score follower. Such scoring has been featured in concert situations such as pieces by composer Philippe Manoury realised using *Antescofo* [137]. Figure 3.6 shows the treatment of static jumps in the action view, which would be similarly handled in the staff view (section 3.3.2).

The second type is where the score elements and their connections are dynamically generated, such as in the work of composer Jason Freeman [88]. In this case, similarly to the dynamic `@target`s from section 3.2.4, we are dealing with an attribute, this time that of an event. For now, we choose to print out the algorithm for jump connection creation in a mouse-over popup, since its runtime evaluation can be impossible to predict.

3.3 Timeline-based Models

In the following, we put forward visualisation solutions for the dynamic constructs from section 3.2, anchored to the linear timeline. Since so much compositional activity relates to timelines, be they on paper or in software, it makes sense to push the boundaries of this paradigm in the context of mixed music.

3.3.1 Representing Dynamics in the Action View

The main challenge in displaying dynamic delay segments alongside static ones is maintaining horizontal coherence. Dynamic sections must be clearly delimited and their consequences shown. To this end we introduce *relative timelines*: once an action is behind a dynamic delay, it no longer synchronizes with actions on the main timeline; rather, the action lives on a new timeframe, which originates at the end of the dynamic delay.

To avoid clutter, a relative time ruler appears only upon focus on a dynamically delayed section. Also, we add a shaded time-offset to depict the delay, as seen in figure 3.7. Since by definition their actual duration is unpredictable, all such shaded regions will have the same default width.

These concepts apply to the display of curves as well. As discussed in section 3.2.1, dynamic breakpoint heights now come into play. Our solution is to randomly generate

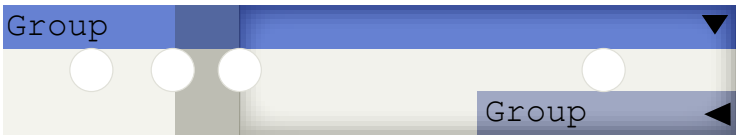


FIGURE 3.7: A group with a dynamic delay between its second and third atomic actions. The subsequent action and subgroup belong to a relative timeline, whose ruler is hidden.

the vertical coordinate of such points, and to mark their position with a vertical shaded bar, as in figure 3.8.

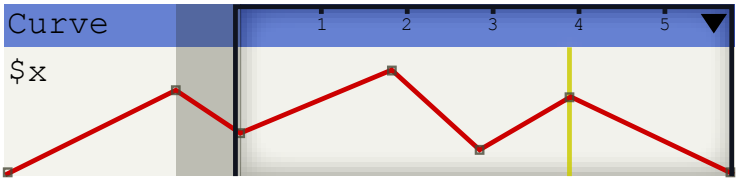


FIGURE 3.8: A curve with a dynamic delay between its second and third breakpoints. The 6th breakpoint has a dynamic value. The relative timeline ruler is drawn.

In our investigations with the core user group at IRCAM, a need was identified for the possibility of "local execution" of a dynamic section, to be able to compare a potential realisation of the dynamics with their neighbouring actions and events. To this end, we propose a *simulate* function for any dynamic block, which transforms it into a classic static group, eliminating its relative timeline. The process makes a best-possible guess for each particular situation, in the context of an ideal execution of the score[6], and can be undone or regenerated. See figure 3.9 for an example of such a local simulation result. The underlying mechanisms are part of the *Antescofo* offline engine, similarly to the full simulation model in section 3.3.2.

FIGURE 3.9: A best-guess simulation of the previously shown curve. The dynamic delay and value have both been actuated.

For the constructs involving a number of iterations over a set of actions, we propose a specific striped shading of the block background, as well as a model for depicting the group's lifespan along the timeline. We chose vertical background stripes for loops and horizontal ones for forall s, according to their sequential or simultaneous nature in

[6]Details on how such ad'hoc trace generation and execution is accomplished can be found in [159].

standard usage, respectively[7]. For the activation intervals of the constructs, we distinguish three situations with their respective models, depicted in figure 3.10: (1) a *definite* lifespan, when the duration is statically known, (2) a *dynamic, finite* lifespan for dynamically determined endpoints, and the (3) *dynamic, infinite* lifespan for activities that carry on indefinitely. These graphic elements are all demonstrated in the examples in section 4.1.6, figures 3.19a and 3.20a.

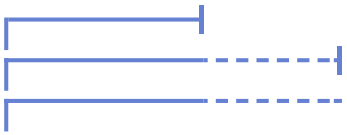


FIGURE 3.10: Definite lifespan (top).
Dynamic finite lifespan (mid). Dynamic infinite lifespan (bottom).

3.3.2 Tracing Performance Simulations

From its conception, *AscoGraph* has included an experimental *simulation mode* that "prints" the whole piece to a virtual execution trace [46]. Much like a traditional score, electronic action *staves* would mark the firing of messages or evolution of continuous value streams along a common timeline. We now present a perfected simulation model, to be implemented into the next version of *AscoGraph*, that more robustly handles the dynamic aspects of the score language and also supports the recently developed *Antescofo* test framework [159].

The general aim is to produce the equivalent to a manually notated score, to be used as a reference for performance and analysis, as well as a tool for finding bugs and making decisions during the composition phase.

Our main design inspiration is a common type of staff notation of electroacoustic music [222], as exemplified in figure 3.11. The standard acoustic score is complemented by electronic action staves, along which the development of computerised processes is traced.

The new display model accommodates all concepts introduced so far: atomic values, curves, action groups and their lifespans. Dynamic values and durations are still indicated specifically; this time we use dotted lines, as the following examples will show.

[7] Of course, a loop can be made simultaneous through a zero repeat period, while a forall can function sequentially by way of branch-dependant delays.

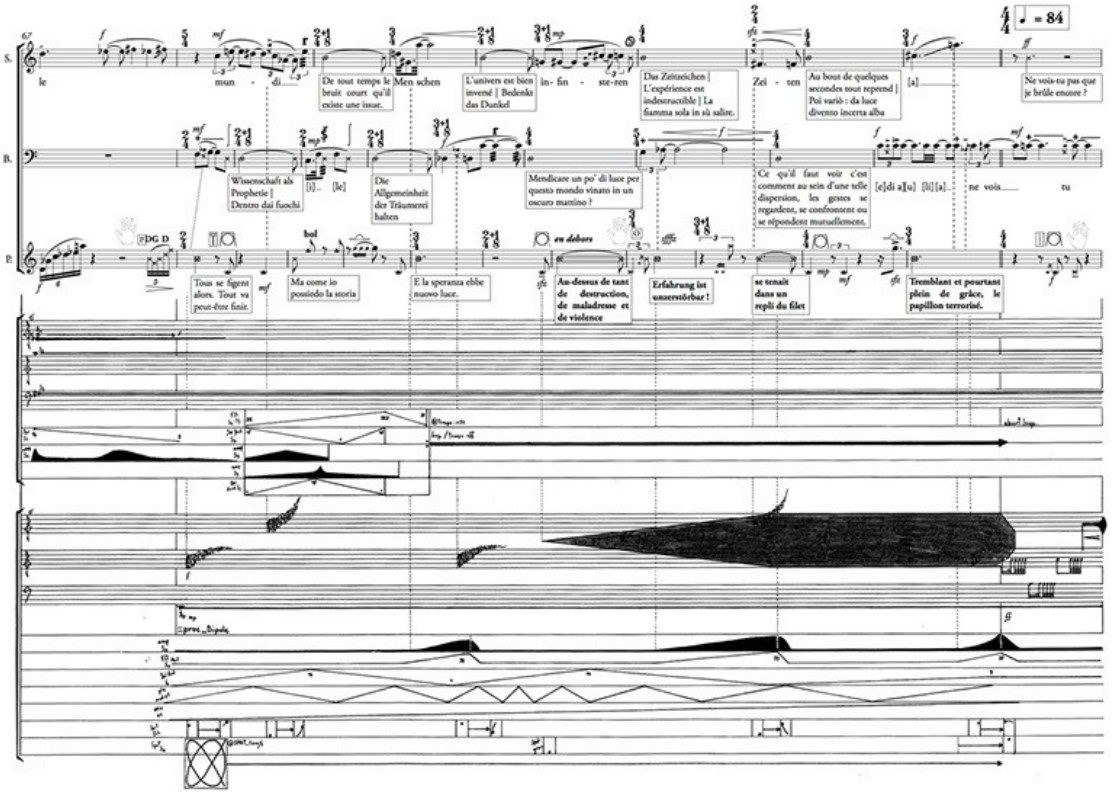


FIGURE 3.11: Electroacoustic staff notation: *Nachleben* (excerpt) by Julia Blondeau.

Horizontally, distances still correspond to musical time, but, as was the case of the shaded areas in section 3.3.1, the dotted lines representing dynamic durations produce disruptions from the main timeline.

Electronic action staves can be collapsed to a "closed" state to save space, where all vertical information is hidden and all components are reduced to their lifespans.

Defining staves

Unlike the action view (see sections 2.5.5 and 3.3.1), in the simulation mode the focus is on reflecting the score's output, not its code structure. While the action view is agnostic with regard to the content of the coded actions, the simulation mode is closely linked to electronic action semantics. Thus, it is likely that commands from disparate areas in the code will belong on the same horizontal staff.

In this simulation trace model, staff distribution is closely linked to the *Antescofo tracks* that are defined in the score, using the `@track_def` command.

```
@track_def track::T {
  "level*"
}
```

LISTING 3.5: Track definition.

In the example in code listing 3.5, the track T contains all score groups or actions whose label or target start with the prefix level, and their children recursively. The model will attempt to print all the corresponding actions to a single staff. Should this prove impossible without creating overlaps, the following actions will be taken, in order, until a "clean" layout is obtained:

1. collapse overlapping action groups to their lifespan segments. These can then be expanded, creating a new sub-staff underneath or above the main one.
2. order the open action groups and curves by relative timeline (see section 3.3.1), and move them to sub-staves as needed.
3. order the open action groups and curves by lifespan length, and move them to sub-staves as needed.

If track definitions are missing from the score, the staff configuration will simply mirror the action group hierarchy. Figure 3.12 shows a possible reflection of the group from figure 3.7, whose corresponding score is Group g1 from code listing 3.6 below. The height of a point on a staff is proportional to the atomic action value, according to its receiver[8]. It is possible to have several receivers on a single staff, each with its own height scale (as in section 3.5.1), or with common scaling (as in the present section).

FIGURE 3.12: Staff representation of a group containing a dynamic delay and a sub-group.

Since the same item can belong to several tracks, this will be reflected in its staff representation. By default, a *primary* staff for the item will be selected, and on the remaining staves the item will only be represented by its lifespan. The user can then expand/collapse any of the instances. The primary staff is selected by the following criteria:

[8]Recall the general atomic action code syntax: [<delay>] <receiver_name> <value>. A receiver might get no numeric value, or a list of values. We use the first value if any, or else a default height of 1.

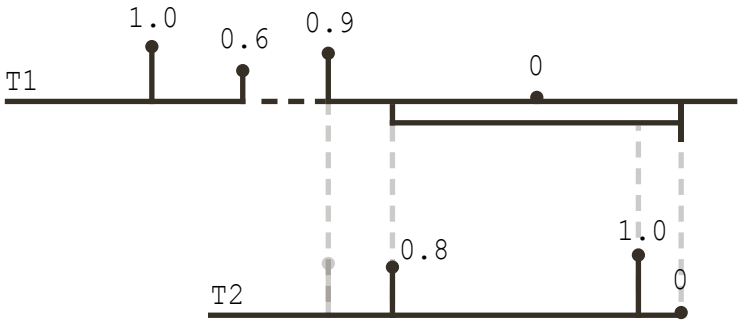


FIGURE 3.13: The elements of T2 are also part of T1 (in collapsed form).

1. least amount of overlapping items
2. smallest distance to staff's track definition: an identical label match will be a closer match than a partial match, which is closer than a parent-child group relationship.

In figure 3.13 we show the same group from figures 3.7 and 3.12, now with two tracks defined: T1 for the main group and T2 for the subgroup. The subgroup and its contents also fall under the track T1 definition, which is why the subgroup lifespan is represented on the T1 staff.

Note that, while the subgroup's timeline starts simultaneously with the m3 0.9 atomic action, its first triggering is m21 0.8, which comes after a .4 beat delay. Since, as we have explained, the simulation view focuses on reflecting the *execution* of the score, the subgroup lifespan on the T1 track is represented as starting with the m21 0.8 event.

Similarly to the action view (section 3.2.3), whenever blocks are represented by their lifespans. However, here the user can expand the contents of the whenever block on a new staff, marked as being out-of-time – much like populating the auxiliary inspector panel, as we describe in section 3.4.2. We present a practical approach to visualising whenever constructs as part of the example in section 3.5.1.

An alternative to the automatic layout generation function is adding tracks one by one using context menu commands. Naturally, users will be able to employ a mix of both strategies, adding and removing score elements or entire tracks or staves to create a desired layout.

While some layouts produced with this model might prove satisfactory for direct reproduction as a final score, we will also provide a vector graphics export function, where a composer can subsequently edit the notation in an external program. Finally, the layout

```
; [PREVIOUS EVENTS]
NOTE C4 0.8 e2 ; (length:
    0.8 beats. label: e2)
Group g1 {
  0.5 m1 1.0
  0.5 m2 0.6
  $x m3 0.9
  Group g2 {
    0.4 m21 0.8
    1.2 m22 1.0
    0.3 m23 0
  }
  0.8 m4 0
}
NOTE 0 2.7 e3 ; silence
NOTE D4 0.8 e4
Curve A @Action := print
    $x {
  $x {       {1.0}
    0.4 {0.3}
    0.2 {0.7}
    0.2 {0.0}
  }
}
NOTE E4 1.0 e5
m5 0.9 @local
```

LISTING 3.6: Score for figures 3.14, 3.15.

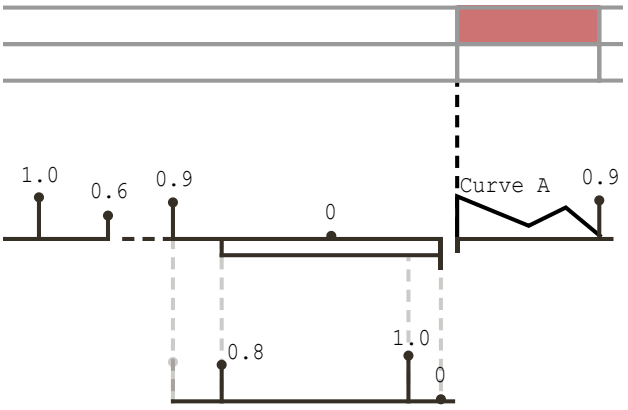


FIGURE 3.14: Synchronising to an event: the piano roll is focused on the D4 note which triggers Curve A.

can be saved in an XML format and included alongside the *Antescofo* score file of the piece.

Representing sync points

We showed in figure 3.13 how the start and end of a subgroup's relative timeline (coinciding with the actions m3, valued at 0.9, and m23, valued at 0) are marked by vertical shaded dotted lines. Similarly we signify a return to the main timeline, by synchronising to a certain event, as in figure 3.14, where curve A is no longer part of the relative timeline before it; it synchronises to an event, depicted by the red rectangle on the piano roll. The corresponding *Antescofo* score is presented in code listing 3.6.

We take a similar approach for dynamic synchronisation targets, as exemplified by the case study in section 3.5.2. Again the sync relationship will be represented by a dotted line, this time parallel to the timeline.

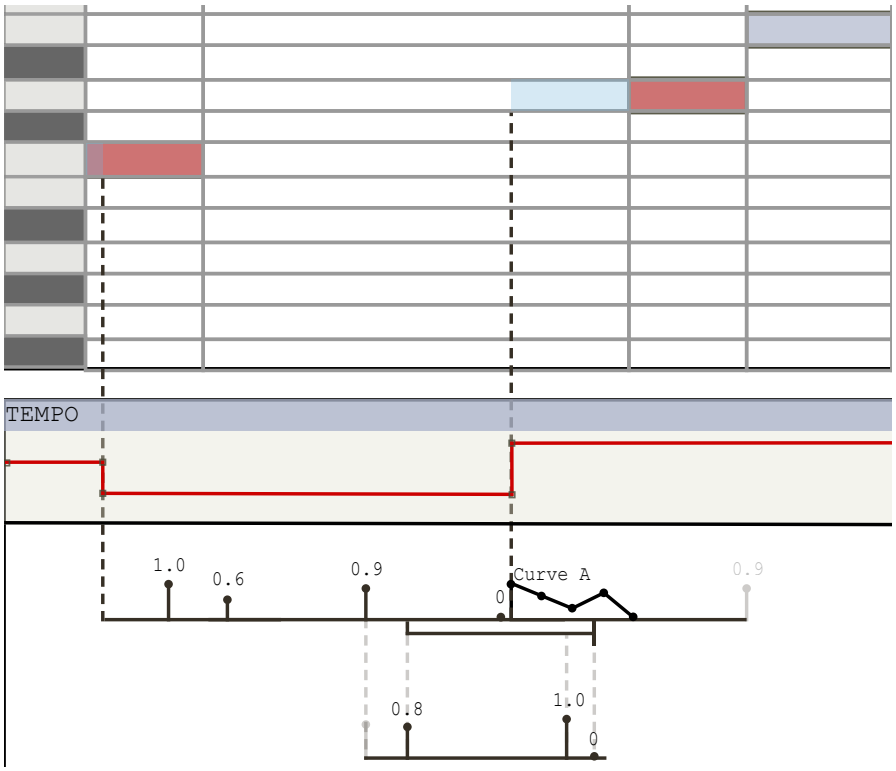


FIGURE 3.15: Partial test scenario: the event e2 (C note) is .125 beats late, the event e4 (D) is .75 beats early, the event e5 (E) is missed. The curve is quantised into 5 actions: c1−c5.

Visualising test traces

Beyond the *ideal* trace produced by executing a score with all events and actions occurring as scripted in the score, the simulation view extends to support the Model-Based Testing workflow [159], which builds a test case by receiving a timed input trace, executing the piece accordingly and outputting a verdict. Such an input trace describes the behaviour of the listening machine, by way of the deviations of the detected musician activity from the ideal score. For each deviation, *Antescofo* computes a new local tempo, based on the last detected note duration.

We propose an example partial test scenario in Table 3.1[9], again corresponding to code listing 3.6. Since the last line of code is an atomic action @local-ly synced to NOTE E4, and in our example input trace (see code listing 3.7) the event is missed, then the action will remain untriggered.

[9]This table and the corresponding trace listings are a sample of the automatically generated output of the *Antescofo* Test framework.

	Musician		*Antescofo* estimation		Event durations		
cue	timestamp	tempo	timestamp [s]	tempo	real	*A.* estimate	Relative duration
e1	0.0s	90.7	0.0	102	0.661s	0.588s	1.125beats [long]
e2	0.661s	115.4	0.58 zzz 0.66	90.7	1.819s	2.315s	2.75beats [short]
e4	2.481s	N/A	!2.481	115.4	irrelevant		
e5	[missed]	N/A					

TABLE 3.1: Partial test scenario. In *Antescofo*'s listening estimation, "zzz" denotes the wait for a late event detection, and "!" is a surprise detection of an early event. The real tempo and duration of event e4 are irrelevant, since the following event is missed.

Timed input traces, as lists of event symbols and their corresponding timestamp and local tempo, can be loaded from text files and visualised on the piano roll, as in figure 3.15. The time distance between the ideal event and its detected trace is highlighted, and missed events are greyed out. The user is able to edit the input trace on the piano roll and save the modifications into a new text file. The computed *tempo curve* τ connects all the local tempo values and spans the duration of the piece; it is displayed along the timeline.

Output traces are produced by computing physical time from musical time. For instance, the timestamp of event e2 from code listing 3.7 is the result of multiplying its input beat position with the corresponding relative tempo: $t(\mathsf{e2}) = 1.125 \times (60/102)$.

Once the input trace has been processed, the simulation view updates accordingly. The offline engine does a best-effort attempt to produce a veridical realisation of the electronic score. For instance, any whenever blocks are fired just as they would be during a real performance, by monitoring their activation condition. This allows their contents to be displayed on the appropriate staves alongside the regularly timed actions – which would be impossible without a known input trace.

```
IN: <e1, 0.0, 102> <e2, 1.125, 90.7> <e4, 3.875, 115.4>
OUT: <e1, 0.00>  <e2, 0.661> <m1, 0.992> <m2, 1.323> <m3, 1.653>
<m21, 1.918> <m4, 2.183> <e4, 2.481, 60> <c1, 2.481>[1.0]
<c2, 2.585>[0.65] <m22, 2.662> <c3, 2.689>[0.3]
<c4, 2.793>[0.7] <m23, 2.818> <c5, 2.897>[0.0]
```

LISTING 3.7: Test case: example input and output traces. We assume an initial event e1, ideally 1 beat long, with an initial tempo of 102bpm. The input trace format is <label, timestamp(in beats), tempo>. The output trace format is <label, timestamp(in seconds)>[value]. The curve triggerings [c1 ... c5] are determined by keyframe timings and lookup grain.

Effectively, a visualisation of the test's output trace is produced, with any missed actions being greyed out. Any staff layout previously configured by the user is preserved. The corresponding test verdict can be saved as a text file.

3.4 Models Complementary to the Timeline

We have shown so far how the introduction of dynamic processes makes linear timeline-based models partially or wholly inadequate for coherent representation. Along with addressing this issue, alternative models can provide the added benefit of improving focus and offering a fresh perspective on the score.

In this section, we propose two solutions: a tree-based display of the score's hierarchical structure and internal relationships, and an auxiliary panel that focuses on specific, possibly recurring actions or groups.

We note that these two models are currently under development as part of the roadmap towards the next major version of *AscoGraph*. The final implementations may vary slightly from the specifications presented hereon.

3.4.1 The Hierarchy View

There are significant precedents of graphic tree representations for grammars in the computer music literature, such as for Curtis Roads' *TREE* specification language [175].

In a similar way, we can interpret the *Antescofo* language as a Type 2 context-free grammar, and construct the hierarchical tree of a score as follows. The primary nodes of the tree are the instrumental EVENTs. Their siblings are the preceding and following events, vertically aligned. Should a jump from and/or towards one of several events be scripted, then one event node can have several downstream siblings.

ACTIONs are secondary nodes, connected to their respective event nodes in a parent-child relationship. The branch structure of the action nodes mirrors the groupings in the score. We have designed a set of glyphs for all *Antescofo* score elements; see figure 3.16.

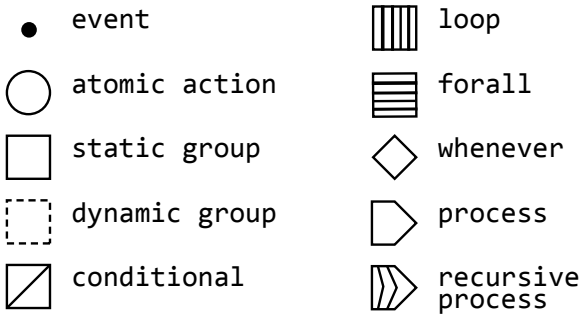


FIGURE 3.16: Glyphs used in the hierarchy tree model.

Aside from the parent-child and sibling relationships defined so far, we also provide ways to indicate internal relationships. These include:

- common variables or macros (colour highlighting);
- common process calls (colour highlighting);
- synchronisation targets (dotted arrow).

The user can selectively activate them permanently, or they can appear upon mouse hover. Figure 3.17 shows an example of all three types of relationships between nodes.

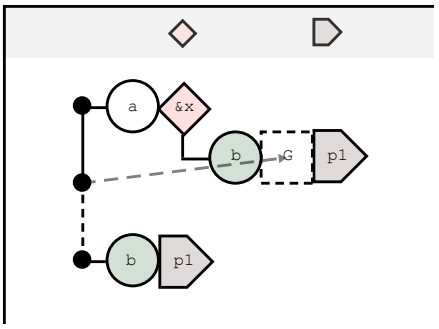


FIGURE 3.17: Example of a hierarchy tree. Group G synchronises to the second event.

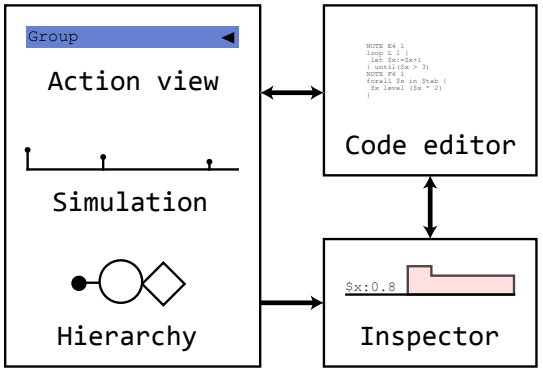


FIGURE 3.18: Explicit linking in *AscoGraph*: the three main views are linked to the code editor, which is linked to the inspector. The main views also allow the user to select items to highlight in the inspector.

To avoid cluttering the tree display, we have decided not to show lifespans in this model. However, whenever and @proc_def nodes are persistently displayed at the top of the window, next to the score tree, for as long as the current zoom view intersects with their lifespan. A click on a whenever node expands its contents in place, and clicking on a @proc_def node expands its currently visible instances within the tree.

3.4.2 The Inspector Panel

In this auxiliary visualisation mode, the user can observe the contents and/or monitor the state of selected groups, actions, or variables. Items can be added via the context menu from the text editor, action view or hierarchy view windows. Once inside the inspector, the item state will synchronise, via a local simulation estimate, with the current position in the score from the other views. This behaviour is consistent with the visualisation principle of explicit linking [181], which is maintained in *AscoGraph* along the diagram in figure 3.18.

The inspector displays a combination between the timeline-based designs from section 3.3. For action groups, we retain the block display (e.g. for showing whenever groups outside the timeline) and we use horizontal staves to visualise the values in variables and action receivers, along with their recent histories. The added value is two-fold: block display of out-of-time constructs, and persistent monitoring of values (even when they have lost focus in the other views).

The hierarchy view and the inspector panel are both depicted in a working situation in the following section; see figure 3.19.

3.5 Case Studies

We present two use-case scenarios highlighting specific dynamic language constructs used in example *Antescofo* pieces. In each example, the instrumental score is minimal (a single event) and the focus is on the electronic actions produced by the machine in response to the instrumental event, and their visualisation using the four models we have described.

3.5.1 Reiteration and Dynamic Occurrence: loop, whenever

The first example is based on the basic rhythm and soundscape tutorials included in the *Antescofo* software package. Code listing 3.8 shows significant portions of an example algorithmic composition score[10] where a group contains two curve and two whenever blocks, whose states control the behaviour of the loop block. To all intents and purposes, the instrumental score is empty, except for "dummy" events for the start and end of the piece: everything happens on the electronic side where *Antescofo* acts as a sequencer. For an analysis of the score's operation, we refer the reader to the tutorial documentation; presently we shall concentrate on the visualisation solutions, as displayed in figures 3.19a, b, c, d.

The action view follows the code structure, and includes the lifespans of the whenever and loop blocks, as introduced in section 3.3.1. Note how these lifespans are all constrained by the duration of the parent group.

The triggering frequency of loop construct is dictated by the evolution of the tempo (curve tempoloop1) and the beat division rate ($cycleloop). Their interdependence is reflected in the simulation view staff display. In the loop, receiver names are drawn by string concatenation via the @command instruction. The layout has been configured to draw a staff for each of the three receivers, containing pan and amplitude pairs.

[10] as found in JBLO_loop_ex-StepTWO.asco.txt.

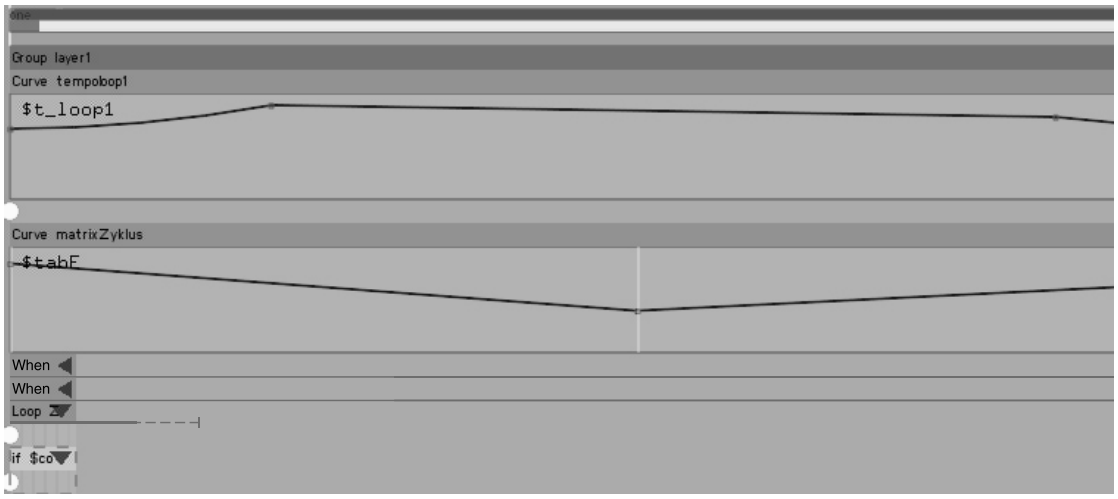
The hierarchy view uses EVENT objects one and three as primary nodes, and the layer1 group as the parent secondary node, from which the subgroups branch out. The existence of common variables between the final loop node and the previous constructs is pointed out through colour highlighting. The two whenever nodes are displayed next to the tree, while their parent node is in view.

Finally, in the inspector panel we track the variables $count, $t_loop1 and $cycleloop, assuming the current view position is after the end of the tempoloop1 curve, which ends on the value 90 for $t_loop1, thus fulfilling the loop's end condition. The user might choose to track a different configuration of receivers in the inspector, depending on their particular focus at a specific time.

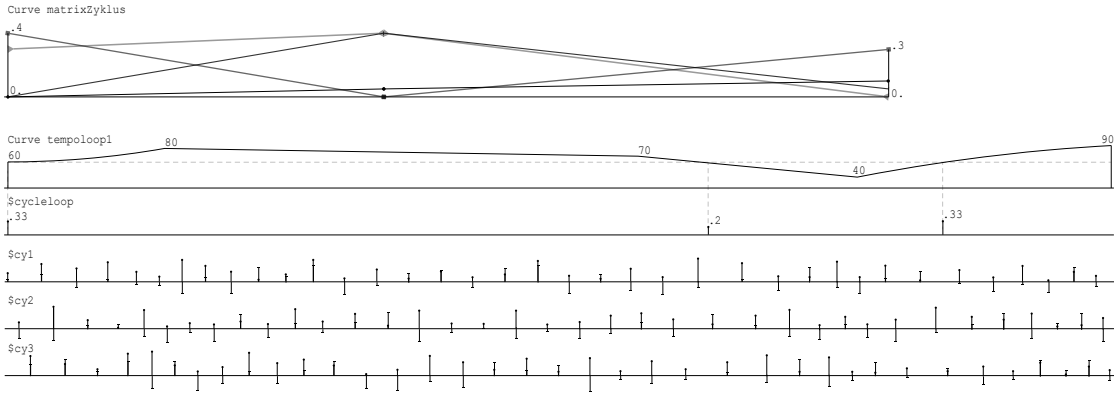
```
EVENT 30 one     ; a dummy score event
GROUP layer1 {
  curve tempoloop1 @grain := 0.05s, @action := recvr_tempo $t_loop1 {
    $t_loop1 {
      ; [STATIC CURVE DEFINITION]
    }
  }
  $count := 1
  curve matrixZyklus @grain := 0.05s {
    $tabE {
      ; [4-DIMENSIONAL CURVE DEFINITION]
    }
  }
  $cycleloop := 1/3
  whenever ($t_loop1 > 60 )
    { let $cycleloop := 1/3 }
  whenever ($t_loop1 < 60 )
    { let $cycleloop := 1/5 }
  loop Zyklus $cycleloop @tempo := $t_loop1 {
    @command(("cy"+$count+"_freq")) ((@random()) + $freqZ)
    @ADSR($count,10,30,90,60)
    @command(("cy"+$count+"_pan")) ((@random()*2)-1)
    if ($count >= 3)
      { let $count := 1 }
    else { let $count := $count + 1 }
  } until ($t_loop1 == 90)
}

EVENT 5 three    ; a dummy score event
```

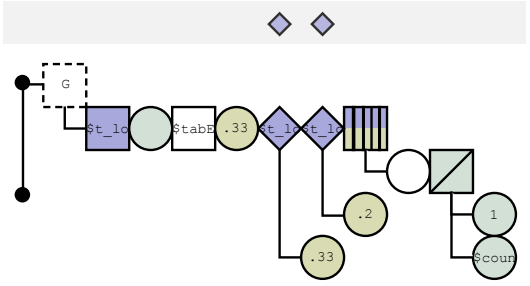
Listing 3.8: Dynamic loop example: the tempo of the loop construct and the delay between iterations are computed at runtime.



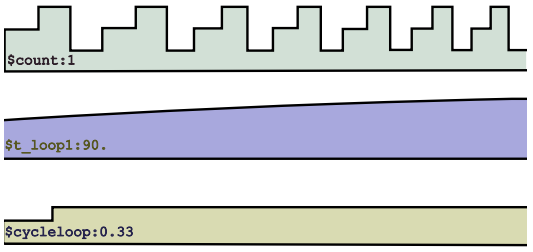
(A) Action view. The right-side part has been cropped for space considerations.



(B) Simulation view



(C) Hierarchy view

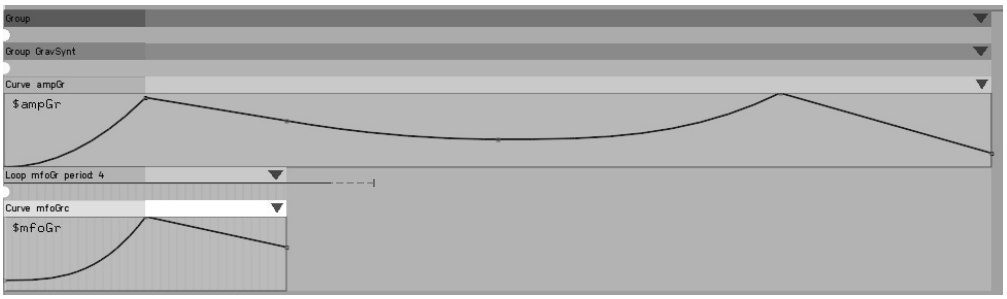


(D) Inspector panel

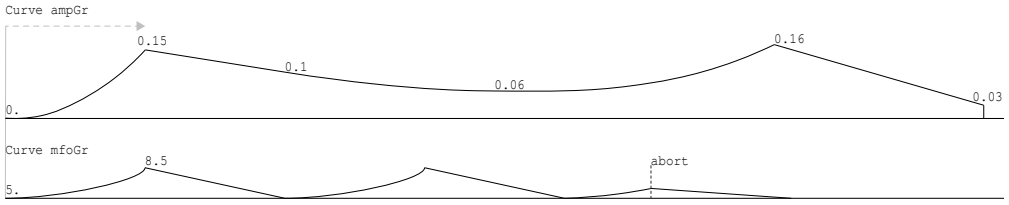
FIGURE 3.19: Visualisations for code listing 3.8.

3.5.2 Score Excerpt: *Tesla*

Our final case study is excerpted from the score of *Tesla ou l'effet d'étrangeté* for viola and live electronics, by composer Julia Blondeau. Again we focus, in code listing 3.9, on the actions associated to a single event in the instrumental part. The visualisation models are proposed in figures 3.20a and b.



(A) Action view. The parent group and all its members have a 2-beat sync target. The loop has a dynamic finite lifespan.



(B) Simulation view. The bottom curve receives an abort message during its third iteration. Both curves inherit the 2-beat synchronisation look-ahead.

FIGURE 3.20: Visualisations for code listing 3.9.

A synthesizer is controlled by the components of GROUP GravSynt, which has a dynamic synchronisation target[11] of 2 beats. We indicate this lookahead interval as a shading of the group header in the action view, and a dotted line arrow in the simulation view.

The group contains the following: a static message to the SPAT1 receiver, a triggering of the ASCOtoCS_SYNTH_Ant process (previously defined through @proc_def), and 5ms later, the triggering of the SYNTH_Ant_curveBat process. After 5ms, curve ampGr is launched, which lasts 14 beats. Simultaneously with the curve above, a loop is triggered, controlling an oscillation in the $mfoGr variable. Each iteration creates a 4-beat curve that starts where the previous one left off (after aborting the previous curve if necessary), and finally, when the loop is stopped, its @abort action ensures a smooth 2-beat reset of $mfoGr to the value 5. Thus, its lifespan is dynamic and finite: it has no set end condition, but is stopped by an abort message elsewhere in the score.

The inconvenience of an unknown endpoint is removed in the simulation view, which by definition is based on an execution of the score. This model is also able to unfold the loop construct in time along its computed duration, attaching the @abort sequence at the end.

[11] as defined in section 3.2.4.

The hierarchical and inspector views provide no new insights in this case; we omit them for space considerations.

```
NOTE D3 4
; [STATIC ATOMIC MESSAGES]

GROUP GravSynt @target [2] {
SPAT1 xy 0 -0.9
::ASCOtoCS_SYNTH_Ant(50,0.,50,0.09,0.,0.00001,0.001,5,1)
0.005s ::SYNTH_Ant_curveBat([0.0001,2,0.15,2,0.0001,4,0.14,4,0.0001])
0.005s curve ampGr @grain := 0.05s,
    @action := ASCOtoCS_SYNTH_L c kampAntes $ampGr {
      $ampGr {
        ; [STATIC CURVE DEFINITION]
      }
    }
$mfoGr := 5.
loop mfoGr 4
  @abort {  ; this is executed at the abort point:
    curve mfoGr @grain := 0.05s,
      @action := ASCOtoCS_SYNTH_L c mfoAntes $mfoGr {
        $mfoGr {
            { $mfoGr }
          2   { 5. }
        }
      }
  } {     ; this is the main loop:
    abort mfoGrc
    curve mfoGrc @grain := 0.05s,
      @action := ASCOtoCS_SYNTH_L c mfoAntes $mfoGr {
        $mfoGr {
            { $mfoGr } @type "cubic"
          2   { 8.5 }
          2   { 5. }
        }
      }
    }
  }
NOTE 0 0
```

Listing 3.9: Score excerpt: Tesla by Julia Blondeau, lines 444 to 483.

3.6 Discussion

The hierarchical visualisation design was evaluated informally over two phases. During the planning and implementation of the new features, we consulted with the internal community of composers and musicians using *Antescofo* at IRCAM. Their suggestions proved useful in guiding our design. In terms of feedback, there was a lot of enthusiasm over the space saving and clarity provided by the new atomic action display mode, whereas appreciation of the difference between stacking algorithms was generally more gradual. This supports the common insight that algorithm complexity is often unrelated to design impact. The second phase started with the announcement of the new features on the public *Antescofo* forum[12]. This allowed the improvements to be also informally validated by the *AscoGraph* user-base.

For the visualisation of dynamic mixed music processes, we proposed four interlinked display models, building towards a framework that can support various compositional and performative approaches to the music. Throughout this project, we have worked under the assumption that a standardised visual notation would be beneficial for current and future practitioners and theorists. Still, the question should be asked: is such a *lingua franca* a necessary, or even desirable goal to pursue? While this is not the time for a definitive assessment, we would like to contribute a point of view based on our experience with *AscoGraph*.

Visualisation models such as the ones we propose have proven useful in practice, at least for a subset of computer music. This includes, but is not limited to, works which employ real-time score following, algorithmic generation, digital sound processing and sampling, recurring electronic actions and/or dynamic feedback between machine and musician. One case where our models might not be as appropriate, would be for instance spectral music, where the impetus is in the evolution of timbre over time, as opposed to discrete pitched events. Even so, the task of extending our models into this direction might prove worthwhile: *Antescofo* is on the way to incorporating audio processing in its engine, making integrated spectral operations a possibility. Other aspects of music-making that our present work is challenged by are spatialisation (or other heavily multi-dimensional processes), improvisation (which the *Antescofo* language approaches through whenever

[12] http://forumnet.ircam.fr/user-groups/antescofo/forum/topic/new-ascograph-display-features-2/

constructs and/or dynamic jumps) and complex synchronisation strategies. Our proposed methods are still far from comprehensively solving these problems.

Over the whole recorded history of music we have seen notation fostering and being in turn shaped by musical creativity. While there will always be artists working outside of general norms (this is as true today as it's ever been, with bespoke visualisations, audio-visual art, live coding and other practices that integrate the visual with the musical piece), in this chapter we aimed to outline a framework for creating and interpreting mixed music which might apply to, and in turn generate, a large set of approaches.

Today's democratisation and diversity of musical practices means we cannot hope to reach a new canonical notation standard. However, this fact also motivates us towards flexible alternatives to the traditional Western staff. The models we have proposed (hierarchical action blocks, traced staff, code tree, inspector) capitalise on the state of the art in computer interface design, maintaining a generality of applicability while enabling specific perspectives into form and material.

Chapter 4

Real-time Coordination of Sonic and Symbolic Material

Music is characterised by continuous information processing and generation. In humans, they manifest by way of memory and expectation [109], into low-level representations of beat, tonality, spatial localisation etc, but also higher-level features such as emotional reactions and predictions. Indeed, Roads makes the claim that "all music is perceived emotionally and romantically" [178].

This reality is further complicated in the case of live music interaction, where human participants produce *intention* and *action*. To the extent that we recognise it, we represent this active information symbolically, that is by means of musical convention: "at the 3rd repeat of X we play Y", or "lock in time with what the Z is doing". X, Y, Z are high-level symbols, or labels, which we can understand in context due to shared ideas of an underlying musical deep structure of laws and mid-level representations [179].

Even to the most passive, indifferent listener, the information described above is never in a fixed state. Rather, anticipation and intention are part of a network of interconnected flows, dynamically informing and updating each other.

Related paradigms are at work in interactive computer music. The real-time coordination of musical listening and action is essential in an interactive system. Our first contribution to this multifaceted problem is a basic online audio alignment and tempo tracking model, that follows a live performance in relation to a known reference. We

then explore some applications of coordination in automatic accompaniment and mixed music situations, before probing into the formal structure of musical constructs. Our contributions in this chapter add to the modes of conceiving and performing mixed music, considering how the better we can automate the low-level machine understanding of music, the more interesting our high-level interactions will be.

4.1 Performance Audio Synchronisation

In studies on tempo and time-shift representation, Honing [106, 107] posits that global tempo curves alone cannot account for the alterations observed in performances of the same material at different speeds. Nevertheless, the majority of work on automatic accompaniment has focused on computing such a curve to drive the warping of a pre-recorded (or sequenced) backing track [6, 157, 171, 186].

Real-time audio-to-audio alignment, either by itself or as part of a score following framework, has been implemented using strategies based on online dynamic time warping (DTW) [6, 69, 128] or particle filtering [73, 146]. The resulting alignment path (as plotted over the t target and h_t reference coordinates, as in figures 2.7, 4.1) matches each frame of the target audio to the reference, but can contain intermittently unnatural slopes. Conversely, algorithms that aim for realistic tempo curves tend to produce higher piecewise onset alignment errors [183, 186], due to the curve fitting produced by smoothing the match path.

To explicitly make alignment paths usable as accompaniment curves, *tempo models* have been introduced to translate the path slope into a realistic *relative tempo* value [6]. In the basic sense, a diagonal orientation equals unity relative tempo, while slope deviations move the tempo up or down.

So far, most audio-to-audio-derived tempo models in the literature have also used onset information from the symbolic score: an offline method that rectifies the path between each onset [149], or models that compute a weighted mean of local tempo over recent onsets [6] or score states [183, 186]. Onset-agnostic models are not as frequent: we cite the basic fixed window measure in [149] and the proportional adaptive model in [157]. Section 4.1.3 adds our contributions to this research area.

In cases of unexpected performance variation or structural differences such as jumps or repeats, online alignment algorithms can temporarily produce erratic local outputs. Alignment-based tempo models can alleviate the problem by smoothing over minor errors, but the larger issue of systematic error[1] detection and correction remains. As [62] and others have indicated, score following is a piece of the machine listening puzzle, but not the whole story.

We propose a system that tackles the issue of errors in music with a strong rhythmic pulse, by tracking tempo in two ways: a model based on audio-to-audio alignment by default, and a beat tracking-based model which takes over when the alignment path becomes unreliable. The resulting machine can drive an accompaniment track based only on audio inputs, without the use of a score or any other symbolic ground truth information. The practical justification is that, for reasons of expediency, lack of access, or incompatibility with the material, a musician might rather record a reference performance of a part instead of plugging in a MIDI or MusicXML symbolic score.

Several studies have addressed the problems of performance variation and error, and of structural differences in the context of music alignment. As we saw in section **??**, in the case of alignment to a symbolic score, the problem of jumps is relatively easily solved by marking points of possible divergence [8, 51, 89]. In the online audio alignment case, structural differences are specifically addressed in [6, 223], which continuously monitor different positions in the score in parallel, to account for jumps. These methods, while performing well for tasks such as real-time page turning and annotation, are however not optimised for automatic accompaniment. There is no mechanism to ensure a musically useful tempo during time periods of incorrect alignment caused by mistakes, jumps, or improvisation.

Meanwhile, beat tracking systems compute tempo curves by detecting the dominant metrical pulse in the live input [182, 202]. As such, they are geared towards scenarios with prominent periodic beats, where the live musician might be locked in with a rhythmic backing track. Wrong notes do not affect the tempo tracking performance, and even when the musician deviates from the rhythmic pattern, beat trackers produce a

[1] We use the term "error" here in the mathematical sense. Certainly, in the name of musical expression, a degree of freedom to deviate from the score is expected—or actively encouraged, depending on the context.

reasonably consistent tempo line. This has placed them at the core of "performance following" [201] tasks such as generative drum accompaniment [182] or tonal performance tracking [201, 208]. Such applications indicate that beat trackers are able to drive a performance forward over periods where the live musician strays from the original reference and the audio alignment struggles to adapt, which is the main insight driving our design.

To be sure, our approach is not wholly unique. We already mentioned symbolic following/accompaniment systems where score position and tempo are explicitly modeled [50, 171]. Related examples include the "coplayer robots" of Otsuka et al [156], which use particle filter-based method to estimate position and tempo, and to switch from synchronisation to beat tracking when the estimation confidence is low. The difference in our system is that it is fully score-agnostic, functioning on no symbolic information. Since we cannot base our modelling on an ideal score representation, we will rely on the reference audio for alignment and beat tracking, while also using the accompaniment audio track in our switching mechanism.

4.1.1 The *rvdtw~* Tracker

We first describe the alignment engine underlying our system. The entire application and its source are available[2] as a Max external object called `rvdtw~` ; to our knowledge, it is the only online audio-to-audio alignment tool for Max to date.

Requirements

In our design, the practical properties required for an online following and accompaniment tool are the following:

1. plug-and-play: we emphasise the ease of setting up. Except optional adjustment of parameters, no further action is necessary other than loading the reference audio to be followed. The system does not require any preliminary training or symbolic description of the material.
2. real-time: the online nature of the program limits the computational complexity of the algorithm, and its input information.

[2] see `https://github.com/RVirmoors/RVdtw-` .

3. flexible: any type of audio material is allowed, be it mono- or polyphonic, percussive, acoustic, electronic, etc. The only condition is the musician's ability to reasonably reproduce the reference.
4. adaptable: the system should respond to changes in tempo or variations in the live audio, and not remain stuck in an erroneous state.

The description from section 2.3.2 implies that the conditions (1) and (2) are met by the classic online DTW [69] model. Item (3) is partially a function of the choice of audio features, and we address item (4) in sections 4.1.3 and 4.1.5.

Choice of audio features

As we surmise from section 2.3.1, different material calls for different feature vectors. In our case we use chromagrams, which have proven their flexibility in audio matching and beat tracking for various styles of music [108, 150, 202]. Specifically, we adopted the implementation from [200], computing 12 chroma values, one for each semitone class. We decided not to add onset-based extensions to the features, because (1) since we don't use a symbolic score, we have no certain specification of reference onsets, (2) robustness in tempo detection was deemed more important than onset alignment accuracy.

We also include an alternate feature vector based on MFCCs. These features are widely used in speech content analysis [121], and have been proven to be a good fit for aligning complex audio sequences [94] where the characteristic to be followed is *timbre*, rather than pitch. This reliance on timbre is also their main shortcoming: if the reference and target sequences have significantly different timbral contours, then the alignment quality suffers [108]. We compute 40 MFCCs according to the procedure in [65], keeping the first 40.

While these two options are coded into our system's internal processing engine, the user also has the option to bypass audio feature processing and use external input features, be they onset-based, CQT-based, pitch information, or any other sequence that is thought to adequately describe the input material.

Updates to the basic online DTW algorithm

Our first change is to remove the path slope constraint, allowing the alignment path to trace along the X or Y axes for as long as necessary. To maintain stability, we compensate by adjusting the local match weighting coefficients to significantly favour

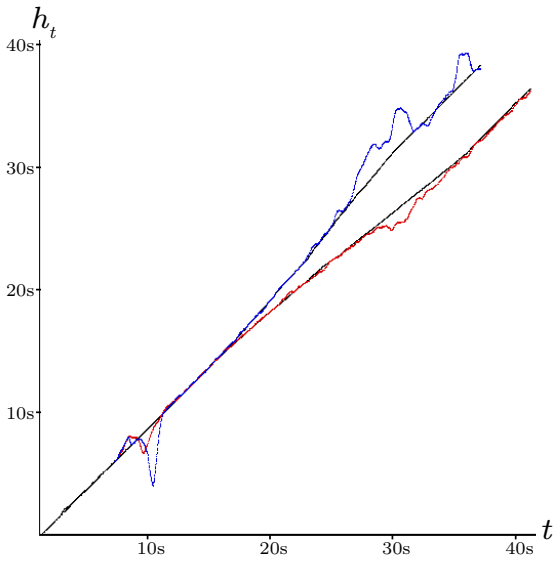


FIGURE 4.1: The online DTW-based audio alignment. Target time t progresses horizontally, reference time h_t vertically. Two test target alignment paths are shown in black. The deviations δ_t^{BACK} from the backwards path are added to each (t, h_t) tuple, in blue for the accelerating target and in red for the decelerating one.

diagonal movement (they default at $w_{\text{diag}} = 1$ and $w_{\text{side}} = 2$), and by adding a constant α to the local cost (as computed in Eqn 2.3), minimising the influence of minor differences between target and reference feature vectors, which could have unpredictably diverted the path:

$$d(i,j) = \sqrt{\sum_k (x_{i,k} - y_{j,k})^2} + \alpha \ , \tag{4.1}$$

Secondly, as inspired by [8], with each new input frame we compute a backwards, offline DTW path starting from the current (t, h_t) position, over a square-shaped match cost matrix covering around 5 seconds in the immediate past. Since both dimensions are now "known", the alignment is considered to be more accurate, and we are able to compute the distance δ_t^{BACK} between coordinates where the main alignment path and the backwards, corrective path reach the border of the window. Whenever this deviation δ_t^{BACK} exceeds a threshold $\epsilon = 50$ms, we adjust the weighting coefficients to favour a path in the respective direction. Figure 4.1 illustrates this behaviour for two test sequences: one gradually slows down, then carries on at 80% tempo before abruptly reverting to the original speed, and the other takes the opposite direction. It becomes evident how the deviation between the main and the backwards path fosters the different tempo regimes.

4.1.2 The oDTW C++ Library

We abstracted the alignment system described thus far (*sans* feature vector extraction) into a C++ library class which might be implemented in other projects. The code is publicly available,[3] along with a basic unit testing suite.[4]

The class provides public methods to initialise the reference memory and populate it, and then to enter target vectors sequentially. The backwards path component can optionally be deactivated. A full specification is provided in appendix C. To our knowledge, ours is the first such library for C++, although similar functionality is present in Dixon's MATCH [70] Java and Vamp[5] implementations[6].

4.1.3 Alignment-based Tempo Models

In our system, the object of the tempo model is for every time frame t to compute a relative tempo τ_t from the alignment path p, so that:

$$\tilde{h}_t = \tilde{h}_{t-1} + \tau_t \ , \tag{4.2}$$

where $\tilde{h}_t \in \mathbb{Q}$ is the *accompaniment coordinate* in the reference series corresponding to the target frame t, and $\tilde{h}_0 = 0$.

Effectively, τ_t acts as a tempo *multiplier* in that it modulates the accompaniment playback speed. For each t, we define the *accompaniment error* ε_t as the difference between the alignment index and the accompaniment coordinate:

$$\varepsilon_t = h_t - \tilde{h}_t \ . \tag{4.3}$$

The appearance of ε_t at a divergence between alignment and accompaniment is exemplified in figure 4.2. We now describe the alignment-based tempo models included in the rvdtw~ tracker. Users can switch between them at any point in the execution.

FW Model

[3] see `https://github.com/RVirmoors/RVdtw-/tree/master/oDTW` .

[4] see `https://github.com/RVirmoors/RVdtw-/tree/master/oDTW_test` .

[5] Vamp is an audio plugin framework, most notably used by the Sonic Visualiser software; see `http://www.vamp-plugins.org` and `http://www.sonicvisualiser.org` .

[6] see `https://code.soundsoftware.ac.uk/projects/match` .

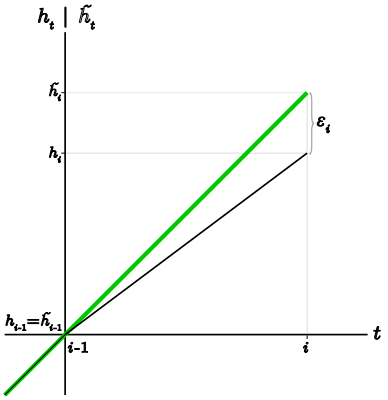


FIGURE 4.2: Divergence between the alignment path and the tempo model. Between the alignment h_i and the accompaniment $\tilde{h}_i$ appears the accompaniment error ε_t.

We start with the fixed window measure, similar to the one in [149], except not centered on the current point since we lack information about future frames. Once every Δt ms, the relative tempo is computed as the slope of the alignment path from the previous breakpoint to the current one:

$$\tau_t^{FW} = \frac{h_t - h_{t-\Delta t}}{\Delta t}. \tag{4.4}$$

The user-adjustable window Δt is set to 500ms by default.

P Model

This model was used by Papiotis in [157]: the fixed window tempo is augmented by a term proportional to the current accompaniment error. This helps the system catch up with the alignment path, in portions where the path takes sharp turns:

$$\tau_t^{P} = \frac{h_t - h_{t-\Delta t}}{\Delta t} + \frac{\varepsilon_t}{K}. \tag{4.5}$$

The constant K needs to be proportional to the window width; in our system we fixed it at $K = 10\Delta t$.

PID Model

From the point of view of control theory [87], we can view the accompaniment driver as a dynamical system producing at every step an error which needs to be minimized over time.

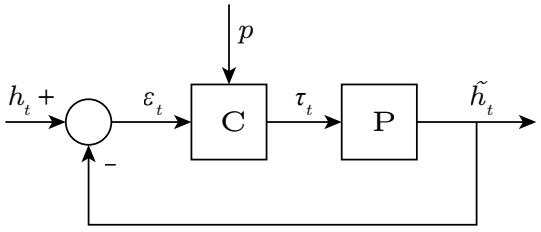


FIGURE 4.3: Block diagram of the PID model's feedback control loop. The controller C receives the accompaniment error ε_t and the alignment path p and emits a new tempo multiplier τ_t, which the plant P uses to compute the accompaniment coordinate $\tilde{h}_t$.

The diagram of this feedback loop is shown in figure 4.3: the P block is the *plant* in control theory terms. It computes the accompaniment index according to Eqn (4.2), which then produces the error which is fed into the controller block C. Moreover, the controller also receives feed-forward information about the current alignment path p—this supplemental feature greatly enhances system stability and accuracy.

For the C block we employ the common PID controller [87], which aims to minimize the error by accounting for its current value (through a proportional term), its accumulated history (an integral term), and its rate of change (a derivative term):

$$u(t) = K_P\ e(t) + K_I \int_0^t e(\tau)d\tau + K_D \frac{de(t)}{dt}. \tag{4.6}$$

We can convert the general model into discrete time:

$$u_d(t) = K_P\ \varepsilon_t + K_I \sum_{i=0}^{t} \varepsilon_i + K_D \frac{\varepsilon_t - \varepsilon_{t-\Delta t}}{\Delta t}, \tag{4.7}$$

and then compute the tempo multiplier:

$$\tau_t^{PID} = \frac{h_t - h_{t-\Delta t} + u_d(t)}{\Delta t}. \tag{4.8}$$

It becomes evident that the P model presented above is a special case where K_I and K_D are zero.

Tuning the K_P, K_I and K_D parameters of the controller is a research task in of itself, especially since we lack a mathematical model for the p path input. In such cases, empirical iterative adjustments are generally employed. We found a good compromise

between stability, response time and anticipation with the following values: $K_P = 17 \times 10^{-3}$, $K_I = 3 \times 10^{-4}$, $K_D = 0.4$.

Pivot Model

The last model we propose leverages the alignment path in order to select two significant *pivots* at every new time frame. First we need to define the *local tempo deviation* measure v_t, as the sum of frame-to-frame tempo deviations over the past 12 frames (around 140ms):

$$v_t = \sum_{i=0}^{11} |\tau^{\mathrm{FW}}_{t-i} - \tau^{\mathrm{FW}}_{t-i-1}|. \tag{4.9}$$

We assume points of low local tempo deviation to be relatively stable and robust synchronization points, due to the fact that the alignment path slope stays fairly constant in their vicinity. We set three selection criteria for our pivots: the first one is within L steps of the current (t, h_t) position, the second is at least K steps away from the first, and their accumulated local tempo deviation $v_{t1} + v_{t2}$ is minimized.

We can solve this problem in $\mathcal{O}(n)$ by way of a dynamic programming algorithm using a double-ended queue, or *deque* [114]. Our method is shown in listing 2.

Algorithm 2 Finding two pivots t_1, t_2 in the past N positions, minimizing $v_{t1} + v_{t2}$. Pivot 1 is within L steps of the origin; pivot 2 is at least K steps away from pivot 1.

```
Min ← VERY_BIG
for i = 0 to N − K do
  if i < L then
    popped ← false
    while not Deq.empty and v_r[i] ≤ v_r[Deq.front] do
      Deq.pop_front()
      popped ← true
    end while
    if popped or Deq.empty then
      Deq.push_front(i)
    end if
  end if
  Sum ← v_r[Deq.front] + v_r[i + K]
  if Sum ≤ Min then
    Min ← Sum
    pivot1 ← Deq.front
    pivot2 ← i + K
  end if
end for
```

Once we know the two pivots, we can define the relative tempo based on the rectified, straight path between the two time points. We also define the *spaced tempo deviation* v_{t_1,t_2} as the distance between the two pivot local tempos:

$$v_{t_1,t_2} = |\tau^{FW}_{t1} - \tau^{FW}_{t2}|. \tag{4.10}$$

We then use v_{t_1,t_2} as a corrective term, to output the relative tempo at time t:

$$\tau^{D}_{t} = \frac{h_{t_1} - h_{t_2}}{t_1 - t_2} + v_{t_1,t_2} sgn(\varepsilon_t). \tag{4.11}$$

In this implementation we set $L = 1$s, $K = 0.5$s. For cases where several points satisfy the pivot selection condition $Sum \leq Min$, we amend the algorithm to average their coordinates into a "meta-pivot".

General Sensitivity

Finally, a *sensitivity* parameter S produces a threshold value ϵ^{S} which the accompaniment error ε_t needs to reach in order to activate the tempo model. For an appropriate sensitivity value, the system ignores minor fluctuations in the alignment path slope and holds the tempo steady at the last computed value. The threshold falls quadratically from one second to zero with the rise of sensitivity from 0 to 1:

$$\epsilon^{\mathrm{S}} = (1 - S)^2 \times 100 \ . \tag{4.12}$$

The impact of the S setting is obvious in figures 4.7 and 4.9, where the time segments with a constant relative tempo have no background shading.

4.1.4 Evaluation of Tempo Models

As this section will show, in order to adequately evaluate an auto accompaniment workflow, the standard score following metrics and datasets do not suffice. We are looking to match a backing track to various instrumental tracks, and judge how well the system fits them together in continuous time, not just at specific keyframes.

We used the Bach10 dataset [74] (consisting of 10 four-part chorale excerpts played by an instrumental quartet, around 30s long each) by rendering the violin MIDI parts into

Target	Model	μ [ms]	σ [ms]	< 100ms	< 200ms	< 250ms
B10	oDTW	49.23	64.42	67.18%		86.20%
	poly2 [183]	-41.18	56.40		88.44%	
	poly4 [183]	-44.15	58.58		90.70%	
fast	oDTW	-2.73	50.08	**91.76%**		**100%**
	PID	-22.67	107.41	71.43%		95.24%
	Pivot	-20.04	94.02	78.57%		96.42%
slow	oDTW	-37.70	39.66	**90.59%**		**100%**
	PID	2030	2401	35.71%		45.24%
	Pivot	2034	2384	36.90%		45.23%

TABLE 4.1: Marker alignment mean error and standard deviation for the Bach10 violin and our two guitar target sequences. Our online DTW-based alignment algorithm produces results comparable to the state of the art, as measured in [183] (2- and 4-voice polyphony). The tempo model outputs, especially for the slow target, reveal why this evaluation method does not reflect accompaniment performance.

audio for use as reference tracks. For the "live" target we used the violin audio files, and we rendered a mix of the remaining 3 MIDI tracks to create accompaniment trios for each piece.

Since existing datasets such as the Bach10 were not produced with our task in mind, we also produced an evaluation set[7] as a possible blueprint for a more comprehensive corpus. The material is a guitar-drum duo playing a minute of mixed-meter rock music at around 130 beats per minute.

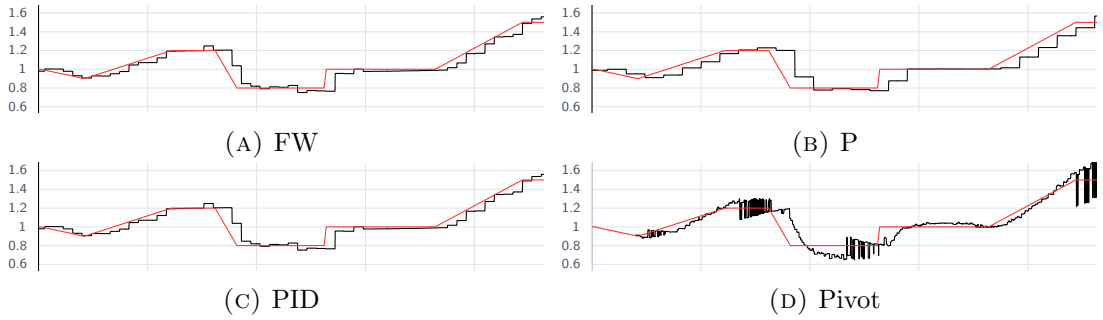
After the initial session we recorded two additional guitar target tracks, as first referenced in section 4.1.1 and figure 4.1: one labeled fast (containing a three-bar acceleration to 120% relative tempo) and one slow (featuring a three-bar slowdown to 80% speed). The system's task is thus to modulate the playback speed of the accompaniment drum track, using the tempo of the target (fast and slow) guitar tracks relative to the reference guitar track from the original session.

Alignment error tests

The overall performance of a system like ours depends on two factors: the accuracy of the audio-to-audio alignment path, and the reliability of the tempo model's output.

To measure the former, we match our fast and slow takes to the reference, computing the mean alignment error [53] and standard deviation over 85 markers placed at every

[7]The audio files used in this evaluation are available at `https://github.com/RVirmoors/RVdtw-/tree/master/_tempo-test`.



(A) FW (B) P

(C) PID (D) Pivot

FIGURE 4.4: Four tempo models (black) reacting to a randomly generated tempo curve (red). Target time t progresses horizontally, relative tempo τ_t is mapped vertically.

downbeat. We also test the violin target and reference tracks from the Bach10 set (totaling 421 markers, one per note). The results in table 4.1 (the "oDTW" rows) show over 90% of errors under 100 ms, amounting to solid alignment performance[8] for the guitar tracks, while the violin alignment is similar to the state of the art [183].

Applying the same procedure to the accompaniment curves produced by our tempo models reveals the inadequacy of this evaluation method. The downbeat alignment error results for the PID and Pivot models in table 4.1 reveal that, as expected, the accompaniment error ε_t introduced by the tempo models causes some shifts in the detection of the downbeat markers. Thus, the audio-to-audio alignment path remains the best solution for precise triggering of events at specific keyframes, such as page turning. However, accuracy comes at a price: the inter-frame movement of the raw alignment path produces unnatural small-scale jumps in the search for the optimal match, and if used directly to drive rhythmic accompaniments such as drum tracks, the results are unacceptable.

Once we have established that our basic alignment is sufficiently accurate, we can focus on the tempo modelling performance. For every frame, we can compute the offset between an ideal ground truth tempo curve and our modelled output. In order to minimise the influence of the alignment engine, we use digitally warped versions of the reference tracks to produce the ground truth,[9] similarly to [149]. As in [149], we compute the average multiplicative difference and standard deviation to obtain percent values of the test-wide error.

[8] The error tolerance threshold depends on the test material and system purpose (e.g. page turning admits a higher error tolerance than accurate sample triggering). For instance, [7] uses a 250 ms threshold applied to complex polyphonic piano and orchestral pieces, for a dynamic score display task.

[9] Digitally warped audio files are much easier to follow accurately than new recordings of the same material.

Model	μ [%]	σ [%]	$< 5\%$	$< 10\%$
FW	**4.10**	6.21	**76.18%**	93.24%
P	6.51	8.14	57.49%	79.80%
PID	4.60	**6.02**	72.91%	**93.58%**
Pivot	6.76	7.06	56.72%	80.53%

TABLE 4.2: Online tempo alignment mean error and standard deviation, averaged over 5 randomly generated tempo warping curves. The PID model is within 10% of the ground truth tempo 93.58% of the time. The Pivot model results are affected by the nature of the experiment. The superior results of the FW model are contradicted by perceptual evidence, pointing to the limitations of this test.

We generated randomly 5 different ground truth relative tempo curves, and ran our models against each one. One such test is displayed in figure 4.4. The tempo alignment error measures are listed in table 4.2. The results are generally worse than those reported in [149] (there the similar FW method produced a mean error $\mu = 2.64\%$ and standard deviation $\sigma = 4.27\%$) but we should keep in mind the following caveats. Firstly, our models run online, which introduces a time lag due to the inability to sample information from the future—this fact is easily observable in figure 4.4. Secondly, by virtually bypassing the audio aligner, we undermined the mechanism of the Pivot model, which leverages alignment quality to determine key sync points.

Most significantly, this quantitative error analysis is not corroborated by the empirical feedback accrued from testing the tempo models in real time. For this reason, we decided not to extend this evaluation method to a wider test database, and instead to conduct a perceptually directed assessment that specifically targets the accompaniment task.

Perceptually oriented tests

It has been shown that the perception of tempo is closely linked to the inter-tap times in tapping exercises [144], and in the context of musical interaction, the individual's abilities for anticipation and adaptation are predictors of their performance in these exercises [142]. Moreover, humans prefer smoothed tempo curves over strict inter-onset-based tempo sequences, which is reinforced by the closer correlation between tapping and smoothed tempo than with raw onset markers [71].

To measure the tempo induction performance of each of our models, we set up the following experiment, using pieces #3 and #5 from the Bach10 set, and tracks fast and slow from our session. Subjects first listened to the original pieces to get familiarized

Target	Model	μ [ms]	σ [ms]	sync
Bach #3	FW	-130.62	118.38	46.61%
	P	-98.53	136.49	68.02%
	PID	**-41.44**	**63.86**	89.39%
	Pivot	-47.06	72.59	**91.30%**
Bach #5	FW	140.12	98.66	40.03%
	P	124.54	102.93	45.58%
	PID	38.92	**56.59**	77.24%
	Pivot	**-17.31**	71.78	**82.48%**
fast	FW	43.45	135.81	76.34%
	P	36.59	141.98	69.11%
	PID	**-13.55**	144.16	81.37%
	Pivot	-22.34	**116.03**	**92.92%**
slow	FW	-154.98	124.96	47.39%
	P	-88.64	135.13	68.21%
	PID	**-32.99**	**80.42**	**92.74%**
	Pivot	-42.02	88.36	**92.73%**

TABLE 4.3: Online tempo tracking: perceptual measures. The two center columns refer to the beat tapping match test. The last column covers the synchronization duration test.

with the material. They were then asked to tap the space bar along to the isolated target tracks, in order to establish a set ground truth sync points. Afterwards the target tracks were used to drive the accompaniment, with the subjects only hearing the live-warped backing tracks and tapping along to approximate the modelled tempo. The error averages are listed in table 4.3, and they were found to provide a more reliable picture of the tempo tracking performance.

We used 7 participants (5 male) between the ages of 24 and 31, M.S. and PhD students with little or no musical training. The models were tested in random order and for each test, we discarded the subject with the highest σ value and averaged the rest. The PID and Pivot models were found to produce the smallest absolute mean errors for all targets. Their relatively low σ values also indicate consistency in their tempo following behaviors.

We also conducted an auxiliary experiment, which might be less quantitatively accurate: we asked each subject to listen to each target track together with its live-warped accompaniment, and to hold down the space bar for as long as the music sounded "in sync". Albeit a more high-level task than the first, this empirical test should still offer some qualitative insights. The results, expressed as the fraction of time that was marked

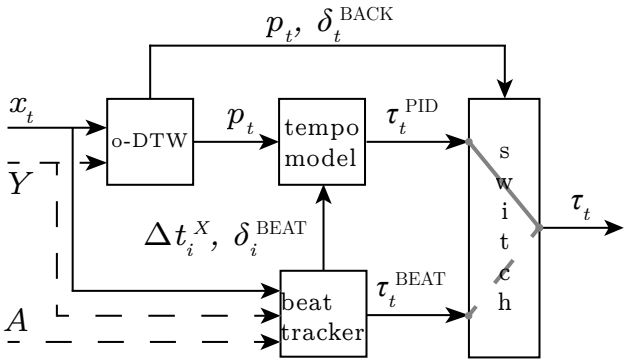


FIGURE 4.5: Framework architecture diagram. Dotted lines signify offline data pre-loading, regular arrows show online data flow. The beat tracker computes a tempo τ_t^{BEAT} based on the x_t input, and sends beat information to the alignment-based tempo model. The choice between τ_t^{BEAT} and τ_t^{PID} for tempo output is based on the current alignment path p_t and its backwards error measure δ_t^{BACK}.

"in sync" are in the last column of table 4.3—again, we discarded the highest σ value before averaging the rest.

4.1.5 Beat Tracking Component

The final component in the rvdtw~ system is a beat tracking module, which we use as a back-up tempo model for when the alignment-based models become unreliable. In this section we are assuming that the PID alignment-based model is active—the same principles would apply to any of the other three options.

We integrated the beat tracker in [202], which is publicly available[10] as a C++ class. The diagram in figure 4.5 shows how the beat tracking module fits into the larger system framework.

We distinguish between the offline phase, where the reference audio to be matched and the accompaniment track are pre-loaded into the beat tracker and their beat positions marked as b_i^Y and, respectively, b_i^A, and the online phase, where the target audio beats b_i^X are detected in real time. Beat durations are measured as:

$$\Delta t_i^{[X,Y,A]} = b_i^{[X,Y,A]} - b_{i-1}^{[X,Y,A]} \quad . \tag{4.13}$$

[10] See `https://github.com/adamstark/BTrack`.

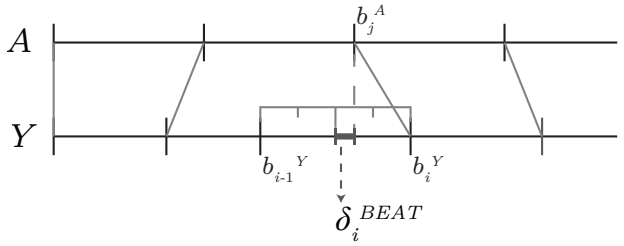


FIGURE 4.6: Computing the distance δ_i^{BEAT} between beats in the reference audio Y and their closest correspondents in the accompaniment track A. In this example, b_i^Y falls closer to an up-beat, so we translate it by $\frac{\Delta t_i^Y}{2}$ towards the closest upbeat b_j^A before measuring the distance.

The Δt_i^X values replace the tempo update interval Δt from Eqns (4.4), (4.5) and (4.8) in real time, ensuring that the FW, P or PID tempo models look at the last detected beat to compute their output: faster tempos need shorter look-backs[11] .

Furthermore in the offline phase, for every reference beat b_i^Y, we compute the distance δ_i^{BEAT} to its corresponding accompaniment beat b_j^A. We take into consideration the possibility of the beats being in reverse phase, so if the distance is larger than a quarter of the current reference beat duration Δt_i^Y, then we translate b_i^Y by half of Δt_i^Y before again computing the distance:

$$\delta_i^{\text{BEAT}} = \min(|b_i^Y - b_j^A|, |b_i^Y \pm \frac{\Delta t_i^Y}{2} - b_j^A|) \ . \tag{4.14}$$

The actuation of the $\pm$ operation depends on whether b_i^Y, being the closest reference beat to b_j^A, comes before or after b_j^A. This process is depicted in figure 4.6, where b_i^Y is translated backwards.

We use this reference-accompaniment beat distance δ_i^{BEAT} as a measure of how rhythmically "locked in" the reference part is to the backing track. This provides an indication of how closely we expect the live target to match the accompaniment beats, which makes it a good modulation factor for the tempo model's S sensitivity parameter. We rewrite Equation (4.12) as follows:

$$\epsilon_i^{\text{S}} = (1 - S)^2 \times 100 + \delta_i^{\text{BEAT}} \ . \tag{4.15}$$

[11] As indicated in e.g. [156], faster tempos can also mean higher following uncertainty, which would point to longer windows being preferable. This aspect is yet to be thoroughly evaluated in our system.

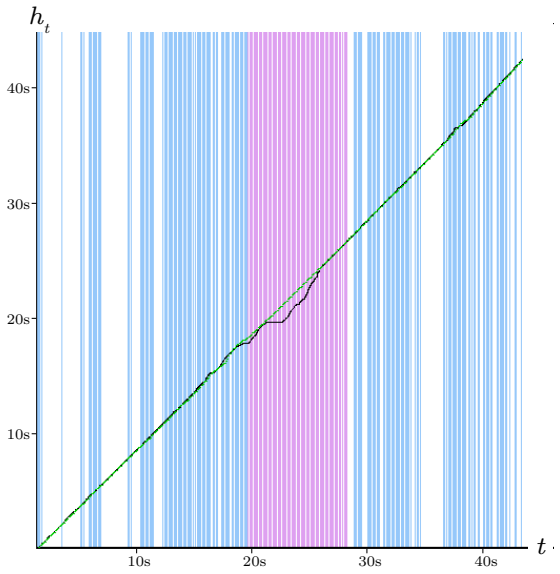


FIGURE 4.7: Online tempo tracking. The black line is the audio-to-audio alignment path; the green line traces the accompaniment coordinates produced by the system. Blue background shading marks time where τ_t^{PID} is in effect; purple marks τ_t^{BEAT} tempo; no background shading marks constant accompaniment tempo.

Now for each new b_i^Y beat reached, the tempo model's activation threshold moves depending on how tightly we expect the target to match the reference. Thus, for "loose" beats we raise the threshold, meaning that small deviations are ignored by the accompaniment and the tempo is kept constant. All our examples were realised using $S = 1$, which would have produced a global zero threshold under Equation (4.12).

The beat tracking module derives a local tempo BPM estimate from the observed beat durations [202]. To produce the relative tempo τ_t^{BEAT} that can drive the accompaniment, we divide the live tempo estimate by the tempo detected at the same beat in the accompaniment track. We define two situations where the system's relative tempo output switches from the alignment-based value τ_t^{PID} to the beat-based one τ_t^{BEAT}:

1. if the tempo produced is more than 3 times slower or faster than the reference, or
2. if the backwards DTW deviation δ_t^{BACK} exceeds a threshold $\epsilon^{\text{B}} = 174$ms, and the difference between the alignment slope[12] $\frac{dh}{dt}$ and τ_t^{BEAT} is greater than $\epsilon^{\text{T}} = 0.15$.

When one of the two conditions is hit, the beat tracker drives the tempo for at least one Δt_i^A beat. Afterwards, if the conditions are inactive, the alignment-based tempo model regains control, and the accompaniment cursor $\tilde{h}_t$ jumps to the respective path position h_t. This entire switching mechanism is laid out as Algorithm 3.

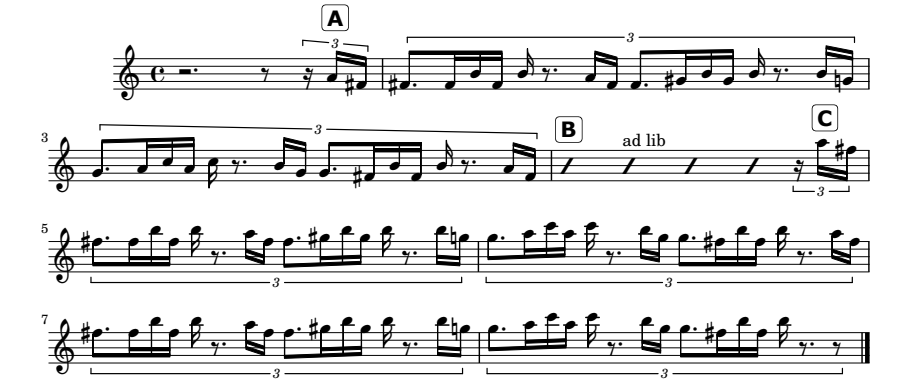
[12] smoothed over the last 40 frames, or 464ms.

Algorithm 3 Switching between the alignment-based tempo model τ_t^{PID} and the beat-based one τ_t^{BEAT}.

$calc \leftarrow$ `NONE`
$waiting \leftarrow 0$
$\tau_0 \leftarrow 1$
for all frames x_t in X **do**
  if $\varepsilon_t > \epsilon_i^{\text{S}}$ **then**
    if $(\delta_t^{\text{BACK}} > \epsilon^{\text{B}}$ **and** $|\frac{dh}{dt} - \tau_t^{\text{BEAT}}| > \epsilon^{\text{T}})$ **or**
    $(\tau_{t-1} \notin (\frac{1}{3}, 3))$ **then**
      $waiting \leftarrow \Delta t_i^A$
    end if
    if $waiting > 0$ **then**
      $\tau_t \leftarrow \tau_t^{\text{BEAT}}$ {*computed by the beat tracker.*}
      $calc \leftarrow$ `BEAT`
    else
      if $calc =$ `BEAT` **then**
        $\tilde{h}_t \leftarrow h_t$ {*jump back to alignment path pos.*}
      end if
      $\tau_t \leftarrow \tau_t^{\text{PID}}$ {*computed in Equation (4.8).*}
      $calc \leftarrow$ `PID`
    end if
  else if $\varepsilon_t \leq 1$ **and** $calc \neq$ `NONE` **then**
    $\tau_t \leftarrow \frac{dh}{dt}$ {*use current path slope.*}
    $calc \leftarrow$ `NONE` {*disable tempo model.*}
  end if
  if $waiting > 0$ **then**
    $waiting \leftarrow waiting - 1$
  end if
  $\tilde{h}_t \leftarrow \tilde{h}_{t-1} + \tau_t$
end for

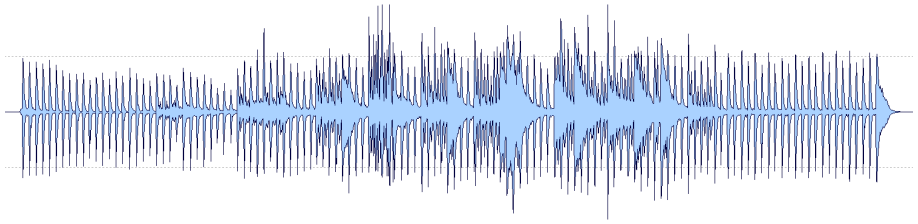
We can follow the algorithm's execution through the example shown in figure 4.7. For the first third of the run-through, the live target closely matches the reference. In the next third, the musician starts improvising, keeping the same tempo but diverging from the original pitches significantly. The $(\delta_t^{\text{BACK}} > \epsilon^{\text{B}}$ **and** $|\frac{dh}{dt} - \tau_t^{\text{BEAT}}| > \epsilon^{\text{T}})$ condition is activated and the tempo is now controlled by the beat tracker, $\tau_t \leftarrow \tau_t^{\text{BEAT}}$. The condition remains active until a few seconds after the musician has resumed playing the scored pitches. By that time, the online DTW path has rejoined the accompaniment path, so the transition back to the original tempo model is seamless.

In the next section we study a more challenging case and test the limits of our proposed accompaniment system.



(A) lead instrument score, unknown by the system

(B) reference track



(C) accompaniment track

FIGURE 4.8: Example scenario. (b) and (c) are pre-loaded into the system.

4.1.6 Case Study

As we noted in section 4.1.4, in order to thoroughly assess the performance of our models, we would require experimental procedures and reference benchmarks that exceed the current standard methods for score following or beat tracking tasks, which mostly measure keyframe-matching ability without specifically studying the musical quality of the resulting accompaniment.

The case for a wider test bench that includes not just several performances of a piece, but also the corresponding accompaniment tracks, is further strengthened by the beat-based tempo model we introduced in section 4.1.5, which strongly relies on backing track information. In the absence of such an evaluation database, a preliminary case study will demonstrate the qualitative improvements over "pure" score followers.

The materials we produced for this example scenario are presented in figure 4.8. A backing track (drums) plays by itself for one measure to cue in the lead instrument

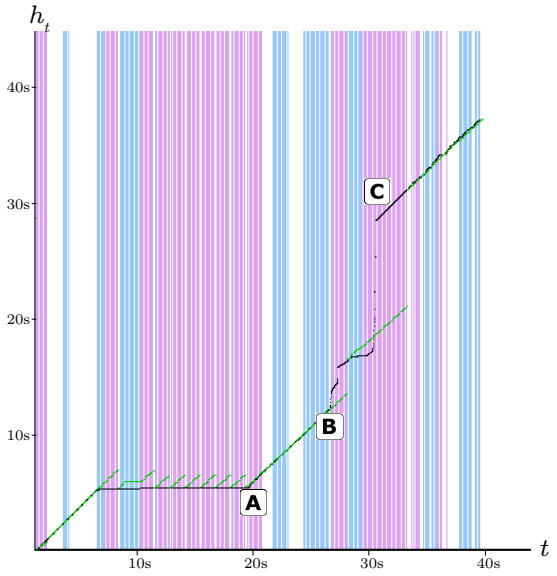


FIGURE 4.9: Example execution of the scenario in figure 4.8. Notations in figure 4.7 apply. See text for interpretation.

(guitar). The two play together for one measure (section A), followed by an improvisation (section B) where the backing track keeps a steady beat. The lead instrument decides when to conclude the improvisation by entering the final two measures (section C).

There are two major questions that our system must answer, without referring to any symbolic information or programmed instructions: what to do when the musician starts improvising, and how to latch back on at the conclusion.

Figure 4.9 shows one realisation of the scenario, where the musician first misses the cue to start playing, which causes the alignment to remain stuck at the start of section A. Other online aligners might produce the same result, but our machine does not stop here: the beat tracker takes control, setting the tempo to 1, which allows the *waiting* variable to elapse during one beat. At the end of this beat, the accompaniment cursor goes back to the baseline, and the process is repeated. Thus, our musician (and the audience) is given a steady beat; once they start playing over it, the alignment-based tempo model regains control.

To model the space for improvisation, we found that filling the space in the reference audio with musical silence (actually the natural background noise of the instrument) gives the desired result. This way, the contrast between (target) activity and (reference) silence makes it instantly obvious when the improvisation begins: the online DTW starts evolving unpredictably, δ_t^{BACK} explodes and the beat tracker takes control.

The realignment in the final measures (as seen in the vertical jump to section C in figure 4.9) depends on one important condition: that the musician pause between the end of the improvisation and the start of the scored coda. This allows the DTW process to match one transition to the other. We found for our particular example[13] a pause of 1.1s to be sufficient, for an online DTW window of 1.49s[14].

4.2 Applications of Audio-to-Symbol Synchronisation

We have already used real-world examples to illustrate notation features in *Antescofo* and *AscoGraph* (sections 2.5.5 and 3.5) and advanced tempo tracking in rvdtw~ (section 4.1.6). We now put forward three more case studies that centrally feature machine-musician synchronisation: automatic accompaniment using rvdtw~ , dynamic accompaniment using *Antescofo*, and a complex IMS mixed music setup again using *Antescofo*.

4.2.1 *rvdtw~* Use Case

We illustrate the typical setup for basic automatic accompaniment via a screen capture of the rvdtw~ tutorial patch in figure 4.10. The first two object outlets reveal the current t and h_t alignment values—these are best used for event triggering that necessitates tight synchronisation. The third outlet is a "dump" destination, outputting among others the $\tilde{h}_t$ accompaniment coordinate. The fourth outlet (not used in this patch) is connected directly to the beat tracker, outputting a bang for every detected beat. Finally, the fifth output sends the relative tempo value τ_t, which can drive the playback by the groove~ object in Max 7.[15]

Once a reference (and, optionally, an accompaniment) have been loaded from audio buffer~s and processed by rvdtw~ , the score (comprising the reference feature vector sequence, annotated with b_i^Y and b_i^A beats) can be saved to a text file for later reuse.

For more systematic triggering and annotation, the object also includes a marker component, similar to *Antescofo* and other audio applications. The marker message followed by a coordinate number saves a marker into the score. From then on, any run-through

[13] The test files are available at `https://github.com/RVirmoors/RVdtw-/tree/master/_smc` .

[14] $c = 128$ frames with a hop of 512 samples, at 44.1kHz sample rate.

[15] Version 7 of Max introduced the @timestretch attribute to its player objects, enabling online time-stretching abilities.

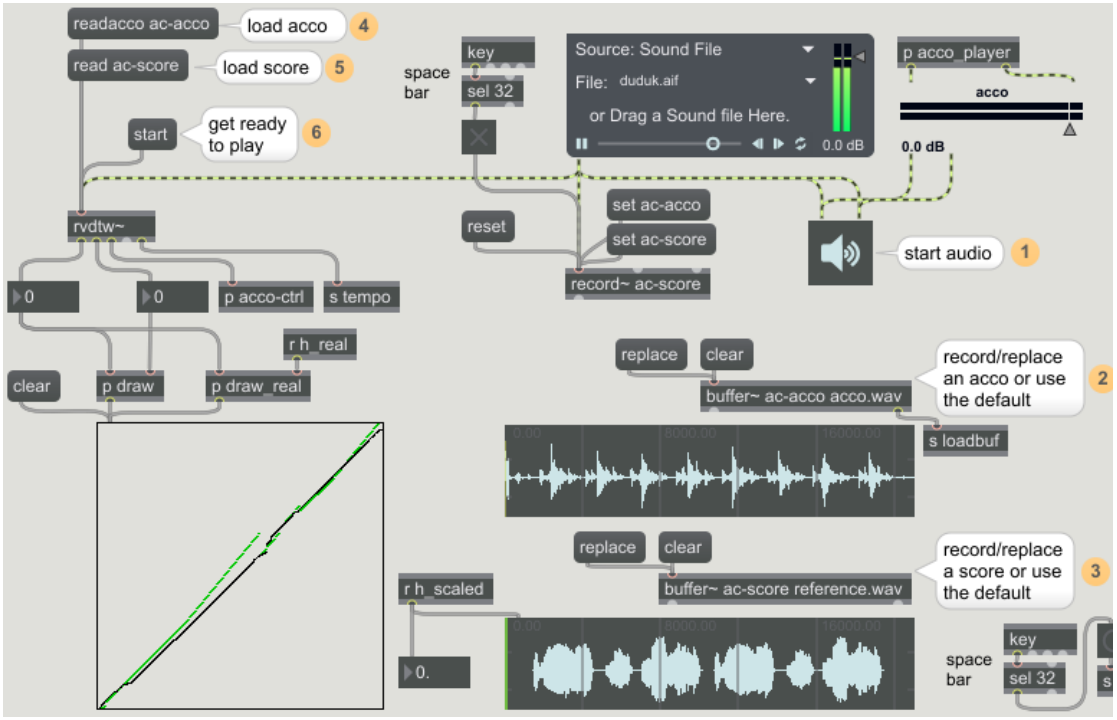


FIGURE 4.10: Basic rvdtw˜ accompaniment setup in Max 7.

will trigger a message through the dump outlet, when that specific marker is hit. This framework can also be applied to an evaluation of marker alignment performance, as we did in section 4.1.4.

For more details, the entire rvdtw˜ inlet and outlet specification is available for reference in Appendix B.

4.2.2 Interactive Drum Machine

We used *Antescofo* at the centre of a Max-based IMS acting mainly as a drummer replacement in a live rock band situation. The computer was set up to receive direct audio input from the electric guitar and bass, and to output a stereo drum track and auxiliary stereo effects. We tested both the rvdtw˜ and the *Antescofo* systems in rehearsal, and while both performed adequately, we decided to pursue a conservative *Antescofo* implementation to maximise stability for the band's first drummer-less show. The setup was conservative in the sense that no tempo tracking was used; instead the system only (1) listens for specific cues from the guitar input, (2) triggers the drum track accompaniment, (3) controls auxiliary processing of the guitar and bass input signals, and

(4) generates a live visualisation based on spectral analysis[16] of the input signals. The accompaniment audio was recorded beforehand with the band's drummer, part of the sessions for a self-produced studio album.

The amount of score following was minimal, for two reasons: the time necessary to encode all the relevant instrumental parts into *Antescofo*, and a reluctance to feed noisy, semi-erratic rock guitar playing into the score follower[17]. This meant that, once the drum track was launched, any markers would be triggered relative to the position reached by the accompaniment. We developed the macro in listing 4.1 to disable the follower and wait for the $sync variable to reach a certain value before restarting it—otherwise, *Antescofo* would simply move on to the next score event without waiting.

```
@macro_def @wait_for($time) {
  print waiting for $time
  antescofo::suivi 0     ; stop follower
  whenever waitfor ($sync >= $time) {
    print have reached $sync
    antescofo::suivi 1   ; start follower
    abort waitfor
  }
}
```

LISTING 4.1: Macro function that turns the score follower off and calls a whenever that waits for the condition to turn it on again.

The $sync value is set periodically from a setvar sync $1 message inside the Max patch, as seen in Figure 4.11. This patch was used for the whole nine-song set; hereon we focus on the track *Case mici*—incidentally, the same that was featured in section 4.1.6.

There are two particular requirements for this song. The first is responding to the guitar intro riff and starting the drum track. Then, around the 35-second mark the song enters a monotonous guitar-drum pattern which must repeat for an arbitrary number of times. We achieve this by generating a random subunitary number and comparing it to 0.97. While the random number is smaller, the backing track repeats the last beat. Once the end condition has been fulfilled, we temporarily change a visualisation tint (vid-color 1)

[16] We used the external objects zsa.mel~ and zsa.flux~ from the zsa descriptors library [134]

[17] To be fair, it's probable that *Antescofo* can handle such a signal quite well, but we weren't ready to take the risk just yet.

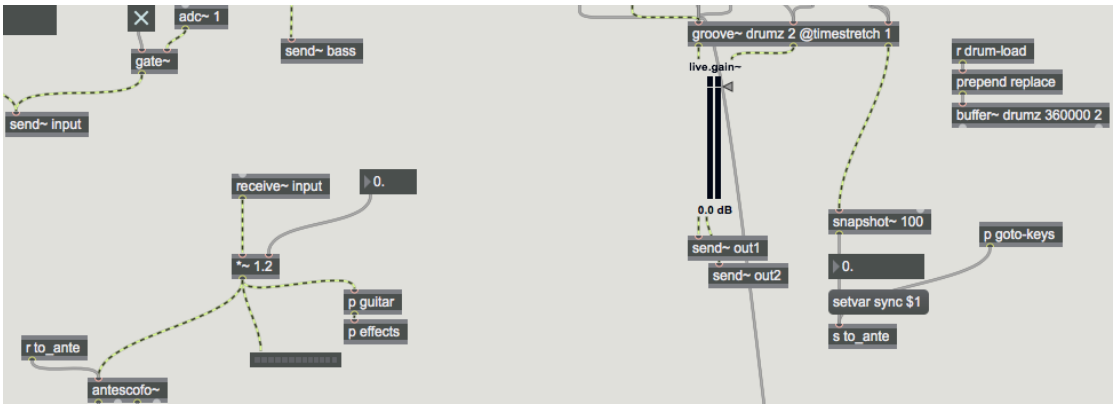


FIGURE 4.11: Fragment of the drum machine Max patch. The setvar sync $1 message is sent every 100ms through the to_ante receiver to the antescofo~ object, updating the internal variable with the current accompaniment position.

to warn the musicians (and perhaps the audience). The entire *Antescofo* score for this song is shown in listing 4.2.

```
BPM 230
@start_song(@casm)   ; init
  macro
NOTE E3 1/2   @hook
vid-color 1
NOTE G#3 1/4
NOTE B3 1/4
NOTE G#3 1/4
NOTE D4 3/4
NOTE E3 1/2
NOTE G#3 1/4
NOTE B3 1/4
NOTE G#3 1/4
NOTE D4 1/4
NOTE D4 1/2
NOTE E3 1/2
NOTE G#3 1/4
NOTE B3 1/4
NOTE G#3 1/4
NOTE D4 3/4
NOTE F3 1/2 @hook
drum-start 6
@wait_for(1)   ; play 'til end

whenever w1 ($sync >= 0.38) {
if ( @random() < 0.97 ) {
print repeat
drum-start 33270
}
else {
vid-color 1
abort w1
}
}

whenever w2 ($sync >= 0.42) {
vid-color 0
abort w2
}
```

LISTING 4.2: Full *Antescofo* score for the song *Case mici*.

FIGURE 4.12: The band *sans* drummer performing *Case mici*. The author can be seen on the left hand side.

A video recording of this performance is available online[18]. A screen capture taken from the video is shown in Figure 4.12.

4.2.3 *Triangolo Aperto*

A more elaborate case study is the computer music design for the piece *Triangolo Aperto* by Fred Popovici. This is an 18-minute, three-part composition for 3 soloists (violin, clarinet and keyboards), orchestra and tape[19], now to be augmented with live electronics. The premiere of this version is planned for 2017.

The role of the IMS is two-fold:

- to align the tape playback to the performance of the soloists and orchestra, as directed by the conductor;
- to dynamically process the signal of the three soloists.

The first task is easily accomplished by having *Antescofo* drive a variable-speed playback device such as SuperVP[20] or groove~. This is achieved by the curve construct in listing 4.3,

[18] see `https://youtu.be/X1V-Z2dNiEo` .

[19] "Tape" is a descriptor inherited from pre-digital new music, where pre-recorded material would be stored and reproduced using analogue tape [103]. The term remains in use today, describing any pre-recorded material that is to be used as an audio source during a performance.

[20] phase vocoder-based time stretching and spectral manipulation tool from IRCAM; see `http://forumnet.ircam.fr/product/supervp-max-en` .

which sends the playback coordinate to the ScrubPos receiver in the Max environment. Over the course of 1611 beats detected by the score follower, the playback head will move from 0. to 402750ms, or almost 7 minutes.[21]

```
curve SVP @target := [2s], @Grain := 0.05 , @Action := ScrubPos $x
    {
      $x {
              { 0. }       ; start
        1611  { 402750. } ; end
      }
    }
```

LISTING 4.3: Driving the variable-speed backing tape.

During the performance, only the three soloists are to be captured via close microphones, which will also provide the material for score following. To prepare for this process, we recorded several takes of the violin, clarinet and piano parts, and went through the iterative process of interpreting the notated score into the *Antescofo* language and verifying that each event is accurately detected.

We faced a major challenge in the case of long pauses (10 seconds or more) between instrumental interventions. Even when these pauses are specified in the score file, the follower has a tendency to come in early and falsely pre-empt the next instrumental event[22]. A possible fix is switching the listening machine to a different source, as seen in the example in figure 4.13 and listing 4.4, where the 25-second gap between violin parts intersects with two clarinet parts. We found that just adding the second clarinet intervention (measures 9 and 10) to the score file was enough to maintain follower stability.

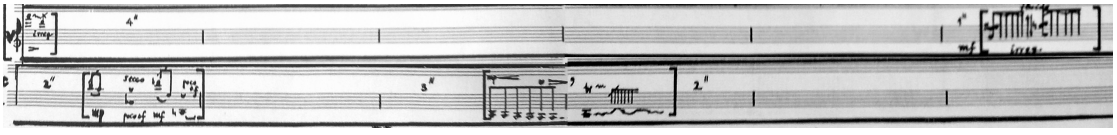


FIGURE 4.13: *Triangolo Aperto* score excerpt. The violin and clarinet solo parts, measures 7 through 12. Each measure is written as 5 seconds.

[21] This is the length of the first of three sections. We handle each section in a separate score file, but they share a common Max parent patch.

[22] This is a known bug in *Antescofo* v0.92, which might be fixed in a future update.

```
TRILL (F7 D7) 4    m7   ; 1s
NOTE 0 48               ; 12s
input-sel 2    ; clarinet
TRILL (A3) 8     m9     ; 2s
NOTE 0  2               ; 3s
TRILL (A3 E4) 10        ; 2.5s
NOTE 0 32               ; 8s
input-sel 1    ; violin
TRILL ( (G4 E5 F#5) (A4 F5 G5) ) 12 m12
  del-time 0
  room reverberance 96
  supervp envtrans 0
```

LISTING 4.4: *Antescofo* score excerpt for figure 4.13. Switching the input source from violin to clarinet and back. The TRILL in measure 12 triggers changes in the electronics.

We now face the second task: designing and programming the live electronics. We start from the following materials:

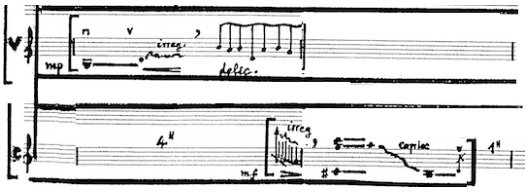
- the instrumental score of the composition;
- a past recording of the piece (*sans* live electronics);
- an audio file of the tape;
- recordings of the violin, clarinet and piano solo parts.

Notably, we lack a recording of the orchestra by itself, since arranging such a session proved impractical. The same can be said about a sampled simulation of the orchestral part: technically possible, but highly impractical. This made us rely more heavily on the tape track, which was conceived as a synthetic "mirror" of the acoustic whole.

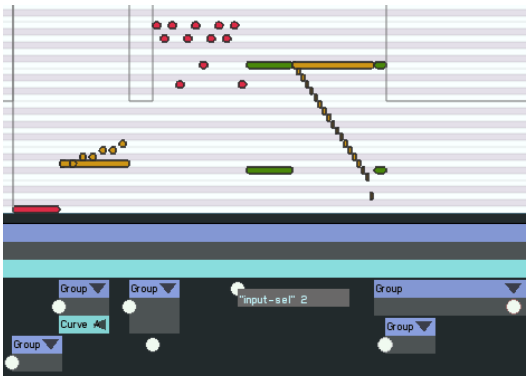
The specifications from the composer were that the soloist interventions act like highlighting "windows" on the tape-orchestra continuum, and it's the role of the live electronics to process the soloist input to highlight this windowing effect, and also smooth out the rough edges between these interventions and the background—the window "frames," as it were.

To this end, we developed a series of processing modules inside the main Max patch:

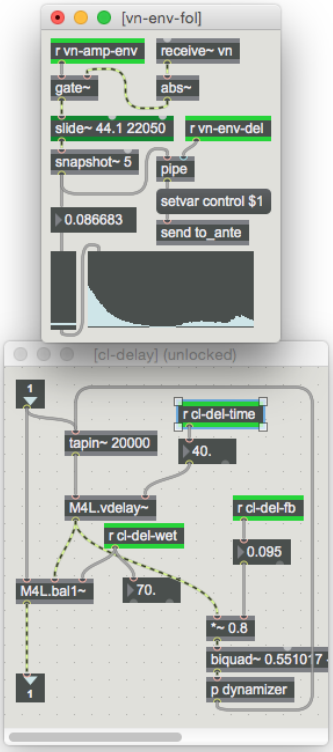
- a pitch shifter and spectral envelope transformer, based on supervp.trans~;
- an echo delay with feedback control;



(A) Fragment of the acoustic score (measures 15 and 16 of the violin and clarinet parts).



(B) Action view visualisation in *AscoGraph*. The mouse is hovering over input−sel 2.



(C) Echo effect implementation in Max 7.

```
NOTE G3 4 m15    ; 1s
vn-amp-env 1  ; activate env. fol.
vn-env-del 16 ; 4s env. delay

MULTI (D4 -> F4) 6   ; 1.5s
; [misc FX actions]

NOTE 0 2          ; 0.5s
; start fb control:
whenever FEEDBACK ($control) {
cl-del-fb $control
}

NOTE B5 2/3
NOTE A5 2/3
; ... [] all other violin NOTEs]
NOTE D5 2/3

input-sel 2    ; clarinet

CHORD (C#4 F5) 4      ; 1s
MULTI (F5 -> A3) 7    ; almost 2s
CHORD (C#4 F5) 1      ; clarinet end

12 abort FEEDBACK ; end fb control
input-sel 1    ; violin
```

(D) Augmented score (edited). The whenever FEEDBACK block sends the delayed control value, and it is aborted 12 beats after the clarinet notes have ended.

FIGURE 4.14: Using Max and *Antescofo* to apply the delayed envelope of the violin audio as a control value for the echo feedback on the clarinet.

- a granular delay, processing several streams of pitch-shifted and time-delayed grains in parallel;
- a spatialisation unit for reverberation and moving sound sources in 2D space, based on spat~[23].

[23] an external package for spatialisation from IRCAM, see `http://forumnet.ircam.fr/product/spat-en` .

These five modules enable us to achieve a multitude of effects, e.g. when the solo violin plays very high artificial harmonics (highlighting the residual high-frequency information produced by the tape and orchestra) we would apply supervp.trans~ to enhance the high end of the solo violin spectrum, and the granular delay and reverberation to create an ambiguous "cloud" that echoes the ambiguity of the high residuals in the tape. Another example would be during a quasi-static midrange granular cluster on the tape track over which the violin and clarinet play semi-percussive interventions (as exemplified in figure 4.13), which we expand and multiply using a variable-time echo delay coupled with subtle harmonisation to give the effect of several closely-intonated instruments playing together.

To control the parameters of the effect modules, we used a combination of (1) static atomic actions, (2) curve constructs evolving over time, and (3) envelope following, e.g. the amplitude envelope of a violin intervention can be mapped to the echo feedback of the subsequent clarinet figure. This latter dynamic appears in figure 4.14.

This relatively limited set of manipulation options already provides a large degree of flexibility. We developed a complete *Antescofo* score for the entire first section of the piece in an iterative manner, trying out different processing combinations for each intervention until we reached a satisfactory whole.

Before carrying on to the rest of the piece, we returned together with the composer to analyse the initial task: producing a live electronics score paired coherently to an acoustic score. We posed the following research question: would it be possible to systematise the process of developing the electronic score, so as to guarantee a degree of coherence? In other words, is there a way to automatically produce a score template that matches the acoustic score, so that for sections 2 and 3 (and potentially different pieces in the future), we would not have to start writing the electronic score from scratch?

Our attempt to formulate such a solution is described in the next section, where we look at using formal grammars as a means to ensure coherence between the acoustic and the electronic scores.

4.3 Towards Generative Grammars for Live Electronics

Formal grammars were introduced by Noam Chomsky as a means to systematically model natural languages [40]. Starting from an *alphabet* and a set of *production rules*, they enable the derivation and prediction of linguistic phenomena that mirror reality. This generative power of formal grammars made them appealing to musicians interested in computer-aided composition. Chomsky defined a hierarchy of grammar types, based on the restrictions imposed on the production rules: from the most general, type 0 or *free*, through type 1 and 2 or *context-sensitive* and *context-free*, to the most limited, type 3 or *finite-state*.

Conventionally, a formal grammar G describing a language $\mathbf{L}(G)$ is defined as the tuple $\{N, T, \Sigma, \mathcal{P}\}$, where:

- T is the *alphabet of terminals*, a set of lowest-level symbols whose concatenations can form *strings*, or expressions. These form the grammar's "surface structure";
- N is the *alphabet of non-terminals*, higher-level symbols representing the "deep structure" of the grammar;
- Σ is the *root token* or starting symbol;
- $\mathcal{P}$ is the set of production rules of the form $\alpha \rightarrow \beta$, where the right side string is generated from and replaces the left side string.

A sequence of productions starting with Σ and ending in a string of terminals is called a *derivation*, and exemplifies the ability of formal grammars to explain and predict all possible sentences in a language. Equating statements with music structures, we note the potential of grammars to describe a compositional style and generate sequences or entire pieces coherent with a set of prescribed laws.

An early study of the powers and limitations of the language metaphor for music composition was provided by Curtis Roads [179]. As a symbolic representation, grammars impose a discretisation of the sound continuum and a compromise in complexity: it is impossible (and impractical) for a formal grammar to fully describe the surface of a musical discourse. This also means that the dynamics of realised music and the resulting cognitive, aesthetic and social implications are beyond the reach of grammars.

These limitations indicate that formal grammars are useful to the composer up to a point: generating and structuring abstract material, which must be then fleshed out and interpreted by humans.

Since the first forays in the 1970s, grammar-based composition systems have reflected the state of computing technology and the research concerns of the time. Roads' *Tree* and *CoTree* languages combined to generate scores based on context-free (type 2) grammars regulated by control procedures [175]. Holtzman's *Generative Grammar Definition Language* (GGDL) was an integrated tool for specifying free (type 0) grammars enriched with "phrase structure" and "transformation" rules to generate scores by experimentation and iteration [105]. Bel and Kippen's *Bol Processor* (BP1 and BP2) evolved from a system for modeling traditional drumming improvisations, to an integrated composition tool supporting metavariables, remote contexts, context-sensitive substitutions and programmed grammars [14]. David Cope's is probably the most elaborate and long-standing research programme, his *Experiments in Music Intelligence* (EMI) database-driven, context-sensitive systems having the ability to formalise and imitate the style of classical composers or original symbolic input [57]. Others have used L-Systems-based grammars, such as Jon McCormack, who added prametric extensions to specify continuous control [141]. Finally we cite the more recent work of Roma and Herrera, who explore graph grammars as a representation of parallelism to support collaborative composition based on a publicly available digital audio database [185].

All of the research cited above provides insights that might prove useful in our endeavour to apply grammars for generating electronic scores. Our present exploration is far from exhaustive; rather it is a preliminary first step towards developing an automatic system that can apply to various compositional styles and material types used in mixed music.

We begin with an approach based on type 2, or context-free grammars, following the recommendations from Roads [179].

4.3.1 Method

We propose a method based on grammar *translation* as a means to generate the abstract form of an electronic action score to accompany an existing acoustic composition,

starting from information about (1) the form of the acoustic score, and (2) the acoustic and electronic material and corresponding processes.

In the 1980s and '90s, grammatical approaches to Western music representation have used the classical *note* (with as few parameters as simply pitch and duration) as the basic unit for an alphabet of terminals [57, 105, 141] reflecting the then limitations of audio processing power. For the purposes of our task, this is insufficient: we are looking for correspondences between electronic and acoustic components, which necessarily occur both on a higher level of abstraction, in terms of compositional structure, and at the lower, sonic level in terms of real-time instantiation.

As will become evident over the course of this experiment, we have found this approach to be especially useful for spectral music [84], where perceptually separate parameters (such as pitch, duration and timbre) are represented as frequencies on a spectrum, to be manipulated coherently over time.

Suppose a grammar $G_A = \{N, T, \Sigma, \mathcal{P}\}$ representing a class of acoustic compositions. We define the set of terminals T as comprised of abstract material and processes which might be generated by a composer from scratch, or reduced from one or several existing pieces. In order to arrive at the written score, the composer would need to instantiate a discourse made up of T tokens by actuating musical parameters and authoring the acoustic piece in the notation of their choice. We emphasise that this is not an automatic procedure: creativity and taste come into play to actualise an abstract structure.

Let us define an equivalent grammar for live electronics, $G_E = \{Q, U, S, \mathcal{R}\}$, representing a class of electronic transformation scores for pieces produced from G_A. As before, the terminals U are a set of materials and processes, which when instantiated, describe transformations of the acoustic activity, generative processes or other kinds of electronic actions.

The coherence between products of these two grammars is achieved by the process of translation, which applies a set of correspondence relationships between the production rules $\mathcal{P}$ and $\mathcal{R}$. These correspondences are not one-to-one: a production rule in $\mathcal{P}$ might have no equivalent in $\mathcal{R}$, or it may have one of several outcomes, depending on some

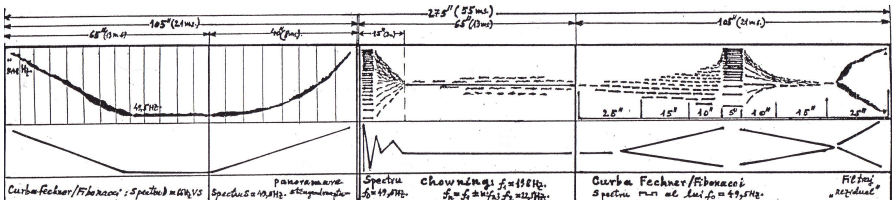


FIGURE 4.15: The generic representation of the piece.

auxiliary procedures. Finally, translation also ensures temporal coherence, by linking the products of equivalent rules in time.[24]

To sum up, in our application the author needs to define two type 2 formal grammars G_A and G_E, and the translation correspondences between them. These produce pairs of discourses in the $\mathbf{L}(G_A)$ and $\mathbf{L}(G_E)$ languages, which can be developed into acoustic and electronic scores, respectively. This entire procedure might involve more work than authoring the electronics step-wise as in section 4.2.3, but the advantages are two-fold: the two discourses are formally coherent, and the composer has access to a flexible framework for generating coherent scores within the same grammatical class.

4.3.2 Case Study

We apply the proposed procedure on a portion of *Triangolo Aperto*, which we introduced in section 4.2.3. Figure 4.15 shows an overview of the piece drawn up by the composer, which can be expanded into the scores to be interpreted by musicians and electronics.

We can extract the following grammar $G_A = \{N, T, \Sigma, \mathcal{P}\}$ from this structural description of the piece, to describe the processes involved down to a desired level of granularity. This is an iterative procedure, and the level of detail depends on the composer's particular purposes. Note the descriptors of "harmonic" and "compressed" spectra, as well as the band-pass filtering processes are just shorthand identifiers used by the composer, and do not have fixed relationships to concrete musical actions. As explained above, it is up to the composer to instantiate such an abstract discourse into an actual musical score. We propose the following specification:

[24] The composer may subsequently decide to produce anticipation or delay effects by way of a specific U terminal. These behaviours would be relative to the baseline established in the translation, where an electronic action terminal is linked to an acoustic terminal.

- $N = \{U, M, H, C, P, F\}$ is the set of non-terminals;
- $T = \{s, h(f), c(f), g(f_i, f_f), e(f), n(f), i, d, a\}$ is the set of terminals, where:
  - s: stationary process
  - $h(f)$: instantiate "harmonic" spectrum with fundamental f
  - $c(f)$: instantiate "compressed" spectrum with central frequency f
  - $g(f_i, f_f)$: glide a band-pass filter from f_i to f_f central frequency
  - $e(f)$: expand the band-pass filter centered on f
  - $n(f)$: narrow the band-pass filter centered on f
  - $i = f_{min}$: the fundamental of a spectrum
  - $d = f_{med}$: the middle frequency
  - $a = f_{max}$: the upper harmonics
- Σ is the starting symbol
- $\mathcal{P}$ is the set of production rules:
  - $\Sigma \rightarrow M\Sigma M \mid U \mid C \mid M \mid H$
  - $U \rightarrow HCH$
  - $M \rightarrow (h(F)c(F))P$: two overlapping spectra and the associated process
  - $H \rightarrow h(F)P$
  - $C \rightarrow c(F)P$
  - $P \rightarrow (PP)$: two parallel processes
  - $P \rightarrow PsP \mid g(FF) \mid e(F) \mid n(F)$
  - $F \rightarrow i \mid d \mid a$

It is now possible to execute the ation described in listing 4.5. Note that the resulting string has a specific level of descriptive capacity, as deemed appropriate by the composer. For instance, materials (spectra) are instantiated and processes (filters) are applied to them, but their end points are not specified. For this particular piece, these are considered implicit; a different composition might have other requirements.

$$\Sigma \rightarrow$$

$$M\Sigma M \rightarrow$$

$$MHCHM \rightarrow$$

$$(h(F)c(F))PHCHM \rightarrow$$

$$(h(F)c(F))PsPHCHM \rightarrow$$

$$(h(F)c(F))g(FF)sPHCHM \rightarrow$$

$$(h(F)c(F))g(FF)sg(FF)HCHM \rightarrow$$

$$(h(F)c(F))g(FF)sg(FF)h(F)PCHM \rightarrow$$

$$(h(F)c(F))g(FF)sg(FF)h(F)n(F)CHM \rightarrow$$

$$(h(F)c(F))g(FF)sg(FF)h(F)n(F)c(F)PHM \rightarrow$$

$$(h(F)c(F))g(FF)sg(FF)h(F)n(F)c(F)PsPHM \rightarrow$$

$$(h(F)c(F))g(FF)sg(FF)h(F)n(F)c(F)e(F)sPHM \rightarrow$$

$$(h(F)c(F))g(FF)sg(FF)h(F)n(F)c(F)e(F)sn(F)HM \rightarrow$$

$$(h(F)c(F))g(FF)sg(FF)h(F)n(F)c(F)e(F)sn(F)h(F)PM \rightarrow$$

$$(h(F)c(F))g(FF)sg(FF)h(F)n(F)c(F)e(F)sn(F)h(F)PsPM \rightarrow$$

$$(h(F)c(F))g(FF)sg(FF)h(F)n(F)c(F)e(F)sn(F)h(F)e(F)sPM \rightarrow$$

$$(h(F)c(F))g(FF)sg(FF)h(F)n(F)c(F)e(F)sn(F)h(F)e(F)sn(F)M \rightarrow$$

$$(h(F)c(F))g(FF)sg(FF)h(F)n(F)c(F)e(F)sn(F)h(F)e(F)sn(F)(h(F)c(F))P \rightarrow$$

$$(h(F)c(F))g(FF)sg(FF)h(F)n(F)c(F)e(F)sn(F)h(F)e(F)sn(F)(h(F)c(F))(PP) \rightarrow$$

$$(h(F)c(F))g(FF)sg(FF)h(F)n(F)c(F)e(F)sn(F)h(F)e(F)sn(F)(h(F)c(F))(g(FF)P) \rightarrow$$

$$(h(F)c(F))g(FF)sg(FF)h(F)n(F)c(F)e(F)sn(F)h(F)e(F)sn(F)(h(F)c(F))(g(FF)g(FF)) \rightarrow$$

$$(h(i)c(F))g(FF)sg(FF)h(F)n(F)c(F)e(F)sn(F)h(F)e(F)sn(F)(h(F)c(F))(g(FF)g(FF)) \rightarrow$$

$$(h(i)c(a))g(FF)sg(FF)h(F)n(F)c(F)e(F)sn(F)h(F)e(F)sn(F)(h(F)c(F))(g(FF)g(FF)) \rightarrow$$

...

$$(h(i)c(a))g(ai)sg(ia)h(i)n(d)c(d)e(d)sn(d)h(i)e(d)sn(d)(h(i)c(a))(g(da)g(dF)) \rightarrow$$

$(h(i)c(a))$: two simultaneous spectra, compressed on f_{max} and harmonic on f_{min}

$g(a, i)$: band-pass filter glides from f_{max} to f_{min}

s : filter remains stationary

$g(i, a)$: band-pass filter glides from f_{min} to f_{max}

$h(i)$: generate harmonic spectrum on f_{min}

$n(d)$: narrowing band-pass filter centred on f_{med}

$c(d)$: generate compressed spectrum centred on f_{med}

$e(d)$: expanding band-pass filter centred on f_{med}

s : filter remains stationary

$n(d)$: narrowing band-pass filter centred on f_{med}

$h(i)$: generate harmonic spectrum on f_{min}

$e(d)$: expanding band-pass filter centred on f_{med}

s : filter remains stationary

$n(d)$: narrowing band-pass filter centred on f_{med}

$(h(i)c(a))$: two simultaneous spectra, compressed on f_{max} and harmonic on f_{min}

$(g(d, a), g(d, i))$: two filters gliding from f_{med} – one to f_{max} and one to f_{min}

LISTING 4.5: Deriving a string of terminals from the starting symbol Σ.

Turning to the grammar for live electronics, $G_E = \{Q, U, S, \mathcal{R}\}$, we can provide the following equivalent specification:

- $Q = \{H, V, I\}$ is the set of non-terminals;
- $U = \{d(t), e(t, f), p(i)\}$ is the set of terminals, where:
  - $d(t)$: delay subsequent action(s)
  - $e(t, f)$: echo effect with delay time t and feedback value f
  - $p(i)$: pitch shift over the interval i
- S is the starting symbol
- $\mathcal{R}$ is the set of production rules:
  - $S \rightarrow VSV \mid H$
  - $H \rightarrow p(I) \mid HVHVH$
  - $V \rightarrow e(I) \mid e(I)V \mid d(I)V \mid VV$
  - $I \rightarrow t \mid t, f \mid i$

This grammar is sufficient for a minimal proof of concept. Following the equivalence to G_A, we might also categorise the terminals into material and process components. The two audio effect types (echo and pitch shift) constitute the material, and the process to which they are subject is the delay unit, which can move them forwards or backwards in time respective to their baselines.

Finally, to establish the temporal baselines and to orient the derivation procedure for G_E strings, we define *translation laws* between members of $\mathcal{P}$ and $\mathcal{R}$, as shown in table 4.4.

ID	$\mathcal{P}$ rule	$\mathcal{R}$ rule
l_1	$\Sigma \rightarrow M\Sigma M$	$S \rightarrow VSV$
l_2	$\Sigma \rightarrow U$	$S \rightarrow H$
l_3	$U \rightarrow HCH$	$H \rightarrow HVHVH$
l_4	$M \rightarrow (h(F)c(F))P$	$V \rightarrow e(I)V$
l_5	$H \rightarrow h(F)P$	$H \rightarrow p(I)$
l_6	$C \rightarrow c(F)P$	$H \rightarrow p(I)$
l_7	$P \rightarrow (PP)$	$V \rightarrow VV$
l_8	$P \rightarrow PsP$	$V \rightarrow d(I)V$
l_9	$P \rightarrow g(FF) \mid e(F) \mid n(F)$	$V \rightarrow e(I)$
l_{10}	$F \rightarrow i \mid d \mid a$	$I \rightarrow t \mid t, f \mid i$

TABLE 4.4: Translation laws from G_A to G_E via the production rules in $\mathcal{P}$ and $\mathcal{R}$.

We arrived at the configuration of production rules and their correspondences by following an iterative, heuristic procedure. Starting from a set of intuitive assumptions (e.g. "compressed" spectra $C \in N$ are analogous to "horizontal" components $H \in Q$)

we developed a set of translation relationships and tested them out on the derivation in listing 4.5. Two factors now help decide whether this configuration is satisfactory: musical intuition (do we have sufficient processes to work with?) and translation completeness (how many of the G_A derivation steps were left without G_E equivalents?). Based on the answers to these questions, we decided whether to revise the configuration and restart the procedure.

As an illustration, we show our final two attempts in listing 4.6. Even using the same set of $\mathcal{R}$ rules and translation relationships, the author still has flexibility deciding which $\mathcal{P}$ rules from the initial derivation to translate, and which to leave out. In the first translation (4.16), we take a greedy left-to-right approach that translates each rule in the same order as the initial derivation. In the second version (4.18), we skip one of the l_9 steps, which allows us instead to apply 4 more productions. The result is a longer string (4.19) that, most importantly, covers the whole temporal domain of the G_A string; whereas the first result (4.17) had to leave several $\mathcal{R}$ rules at the end untranslated.

The acoustic and electronic parameters produced by the l_{10} rules are left to the composer's discretion; optionally, they could also be subject to certain built-in restrictions.

$$l_1, l_2, l_3, l_4, l_8, l_9, \cancel{l_9}, l_5, l_9, l_6, l_8, l_9, \cancel{l_9}, l_5, l_8, l_9, \sout{l_9, l_4, l_7, l_9, l_9}, l_{10}, \ldots, l_{10} \tag{4.16}$$

$$S \to VSV \to \cdots \to e(t,f)d(t)e(t,f)p(i)e(t,f)p(i)d(t)e(t,f)p(i)d(t)e(t,f) \tag{4.17}$$

$$l_1, l_2, l_3, l_4, l_8, l_9, \cancel{l_9}, l_5, l_9, l_6, l_8, l_9, \cancel{l_9}, l_5, l_8, \sout{l_9, l_9}, l_4, l_7, l_9, l_9, l_{10}, \ldots, l_{10} \tag{4.18}$$

$$S \to VSV \to \cdots \to e(t,f)d(t)e(t,f)p(i)e(t,f)p(i)d(t)e(t,f)p(i)d(t)e(t,f)e(t,f)e(t,f) \tag{4.19}$$

LISTING 4.6: Translating the G_A discourse from listing 4.5 into G_E, using the translation laws in table 4.4. Two derivation orders are proposed. Compared to the initial sequence, the productions omitted in the translation are striked through.

We can now trace the baseline equivalences between the components of the G_A and G_E strings, as shown in table 4.5. Note that we are still in the abstract domain, so actual temporal alignments are not automatically imposed.

This concludes our preliminary exploration of using formal grammars to build coherent musical structures in both the acoustic and electronic realms. We discuss several open questions at the end of section 4.4 below.

G_A string (code 4.5)	(hc)	gsg	h	n	c	esn	h	$esn(hc)$	(gg)
G_E string (4.19)	e	de	p	e	p	de	p	de	ee

TABLE 4.5: Baseline correspondences between G_A and G_E terminals. Component parameters have been omitted to simplify the notation.

4.4 Discussion

In the most basic sense, models are "an attempt to describe or capture the world" [191]. By extension, we use tempo modelling to understand and track the perception of tempo, as an expressive performance parameter [93]. We began this chapter with a description of two such understandings: one based on audio following and alignment, and the other on beat tracking. We implemented both models into rvdtw~ , an online system that tracks tempo based solely on audio processing, without reference to any symbolic score data. The two models inform each other, and as such can be considered to form a larger model that includes the switching mechanism which determines which of the two is the active tempo driver.

This framework is geared to a specific scenario, where a musician creates a reference audio track by playing along to a fixed backing track. This accompaniment track is then warped in real time to match a live target, as we showed in section 4.2.1. Such a scenario applies more to popular beat-based music genres than the classical concerto or solo performances that followers such as [6, 171] are designed around. Thus, while the added rigidity serves beat-centred material well, it is less appropriate for strong rubato, where pure alignment-based models perform better. The user can address this by raising the ϵ^{B} and ϵ^{T} thresholds, causing the beat tracker to engage less easily.

For the alignment-based tempo model, we proposed four variants, each aiming to drive an accompaniment in a musical way, by softening the eccentricities and discretisation errors inherent in the alignment path while reacting robustly to tempo nuances in the musician's playing. Our algorithms are all computationally linear, so the associated overhead is minimal compared to the FFT-based feature extraction and audio alignment mechanisms.

The results of the comparative evaluation of the alignment-based tempo models, as well as our in-house testing experience, do not reveal a clear "best" choice for all circumstances. The only across-the-board improvements are shown by the PID model over

the FW and P variants: the addition of an integral and a derivative component greatly contribute to the system's adaptation and, respectively, anticipation abilities. The PID model excels at compensating for the time lag inherent to the online nature of the system. This improved reactivity however comes at the cost of relatively low tempo stability.

In the case of the Pivot model, the interplay between inter-pivot slope and the spaced tempo deviation produce a stable, historically informed output that also triangulates to minimize the accompaniment error. This behaviour of flicking between tempo hypotheses, while musically useful, would however be distracting (as figure 4.4d shows) in a post-factum annotation analysis task, where offline models are superior [149].

Overall, the PID and Pivot models performed very well on the perceptually directed tests. While the method for measuring the duration of synchronicity may not be the most quantitatively accurate, results of 80-90% can justify a positive qualitative appraisal of these models. For the future, we intend to broaden the test database and work towards standardizing perceptual evaluation methods for such real-time musical processing tasks. We have seen that the tempo curve alignment error analysis does provide insight for offline tasks, but can be misleading in the online case, as evidenced by the similar results for FW and PID in table 4.2, when the actual musical performance of the latter was evidently superior.

The second major part of this chapter was dedicated to fleshing out several interactive scenarios that rely on the synchronisation abilities of our rvdtw~ object and the *Antescofo* system. Our main aims were:

- to introduce an automatic accompaniment workflow involving rvdtw~ ;
- to apply *Antescofo* in an unconventional (rock music) setting, where the score follower contribution is minimised in favour of the symbolic synchronisation and triggering abilities afforded by the reactive engine;
- to realise the interactive music principles from section 2.1 via an original mixed music piece.

One insight we can draw at this point refers to the difference in character between symbol-based followers such as *Antescofo* and audio-based trackers like rvdtw~ . We return to Dannenberg's point [62], that different styles of music pertain to certain technological approaches. While initially rvdtw~ might have seemed as neutral a system as

possible (and we maintain the philosophy of trying to deal with any input signal) we found it valuable, while leaving the basic foundation untouched, to build on it towards beat-strong music in order to advance error handling capabilities. Meanwhile, *Antescofo* is a hugely robust *and* flexible system, *as long as* the acoustic score is appropriately formulated. Based on our current experience we can attest that coding and testing *Antescofo* scores is a skill in itself, just as with any programming language.

Our case studies, while fully functional, should be viewed as starting points towards more mature approaches to dynamic interaction. The flexible and modular nature of Max and similar environments is conducive to continual evolution [164].

In the last section of this chapter we proposed one such exploratory path: a method based around the translation between formal grammars as a way to coherently generate an electronic score from an existing acoustic score.

Through the manual processing of a case study score we indicated how our method might be generalised to a wider set of scores. The abstract level of the produced strings means that any implementation will necessarily be more elaborate and dynamic, e.g. realising the $e(t, f)$ terminal symbol into a granular echo effect with a grain size linked to the t time value.

This final step of defining the sonic realisation of each terminal in a grammar is the process called *lexical mapping* [175, 185], which necessarily in our case would have to function differently for the two languages. For the acoustic grammar G_A, in order to capture the sonic reality of the piece, we might employ a hierarchical clustering technique to group discrete sound objects [39], or "atoms" from a recording, into terminals, a process called *grammar induction* [41, 189]. Our limited experiments with extracting atoms using dictionary-based methods (DBMs) [205] confirmed the difficulty of detecting functionally relevant portions of audio and grouping them into clusters that reflect the composer's intention (which is what our grammar's terminals describe); this remains a potential avenue for fruitful future research.

The case of the electronic grammar G_E presents an additional layer of complexity: here we are not dealing with concrete sound objects, but rather with transformation and synthesis processes which respond to incoming audio stream(s) in real time. We are still far from a definitive theory on this issue (although the efforts of Roads [177, 178],

Núñez [153], di Scipio [68] and others [124, 173, 204, 219] offer insightful perspectives and solutions), so in the meantime we leave the task of assigning electronic processes to G_E terminals to the composer/designer.

There are several other open questions which for now only the composer can answer. A piece such as *Triangolo Aperto* (sections 4.2.3, 4.3.2) is based on a general plan which is actuated through several different instruments. Which electronic processes then belong to which instrumental tracks? One indication would be, as with the temporal baselines, to follow the production rules down to the acoustic realisation, but surely this is an ambiguous pursuit in any but the most banal cases.

Another interesting aspect are one-to-many translation laws. In our solution (table 4.4) the laws are mostly one-to-one, with l_9 being many-to-one. One-to-many laws would add to both complexity and flexibility, and might be implemented by conditional (e.g. if , whenever) constructs in a reactive electronic score. These and other avenues are left open to the composer for now, but might be automated or computer-aided in a future application. Its final product: an *Antescofo* score template, complete with acoustic events and corresponding action groups, which the composer/designer can fill in with interaction schemas and parameter values.

Chapter 5

Data Sonification

In previous chapters we worked with representations *of* sound, either to model the synchronisation of sonic and symbolic processes, or to make these processes available for out-of-time observation via visualisation. We now turn to the other side of the coin: representation *through* sound.

In their seminal survey of the field, Kramer et al [116] described sonification as "the transformation of data relations into perceived relations in an acoustic signal for the purposes of facilitating communication or interpretation." We find this definition particularly useful due to the possibilities it leaves open. Communication or interpretation of what? It might be scientific knowledge, or it might be a narrative (that is, music [178]), or art-related experience, or miscellaneous invisible relationships, as we discussed in section 2.4. Best of all (for our purposes), we do not have to choose just one of the above. This quality of data sonification grounds the endeavour we describe in this chapter.

5.1 Real-time Music Generation from EEG Signals

The advent of accessible EEG headsets such as the Emotiv EPOC [76] has made possible the real-time control of live performance environments through brainwave representations. Such a project is *INTER@FAȚA*, an interactive theatre piece centred around the themes of violence and discrimination, where generative musical algorithms are directed by the interplay between a pair of EPOC signal bundles, sourced from a performer and an audience member.

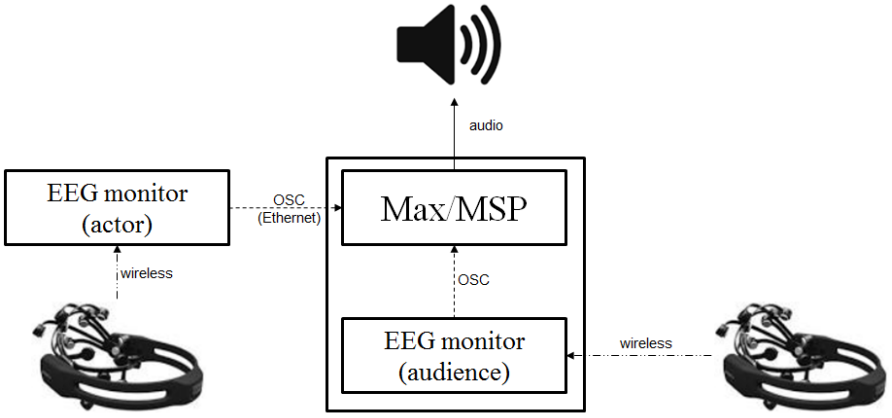


FIGURE 5.1: The system architecture. Two EEG headsets are monitored in real time to control sound generation.

Through the mapping of EEG activity into sound, the audience participates in a non-invasive way in producing the performance. Figure 5.1 shows a diagram of the system architecture, with different members of the cast and audience wearing the two EEG headsets over three portions of the piece, totalling 6 monitored subjects for every performance.

The overall purpose has been to achieve a synergistic relationship between the emotional states and reactions from actors and audience, and the music generated by their dynamics, influencing each other as parts of a coherent whole. While the musical algorithms were developed as part of the workshopping phase of the piece and remained unchanged throughout the theatrical run, the design team did make adjustments to the activation patterns of the algorithms based on the sonic output observed on each performance night.

5.1.1 EEG Data Frames

We developed a bespoke openFrameworks/C++ application[1] that captures the data streams from the Emotiv API[2] and performs spectral analysis on the AF3 and AF4 prefontal cortex sensor output to compute alpha and beta wave activity. The application then relays the combined data frames through OSC to the Max/MSP programming

[1]The application's source code is available at `https://github.com/RVirmoors/eegOSC`.

[2]At the time of development (2014), raw EEG data access was possible through the Emotiv Research SDK. In the meantime, Emotiv have re-branded their SDK and the same features should be available in the Premium SDK. See `http://emotiv.com/developer/`.

environment, where specific parts of the frames control certain musical algorithms at different places in the script.

The input frames thus consist of the following streams: 14 raw EEG amplitude signals from the headset sensors, 5 affective indices based on proprietary Emotiv algorithms, and the difference of valence between channels AF3 and AF4, which has been found to indicate positive or negative emotion [168][3]. Specifically, we compute the valence as the difference between the ratios between the alpha power and beta power in the two EEG signals:

$$v_{AF3-AF4} = \alpha_{AF3}/\beta_{AF3} - \alpha_{AF4}/\beta_{AF4} \ . \tag{5.1}$$

The raw EEG data is transmitted at a rate of 128Hz, while the affective indices and the valence values are computed and sent out at 1Hz. Aside from directly processing real-time data from the EPOC headset, our application can also load EEG data from an EDF or EDF+ file [113] and process and transmit it over OSC for development and testing purposes.

5.1.2 Generation Algorithms

Algorithm control can either be direct (e.g. a proportional relationship between the raw EEG signals from the audience member and a bank of sinusoidal generator amplitudes) or the result of a relationship between different stream trends (e.g. the triggering of a sampling process at the moments where the AF3-AF4 valences of the performer and spectator start increasing in tandem). The musical algorithms are based on continuous control of low (frequency, amplitude, timbre) or high-level (pitch, tempo, duration) musical parameters, acoustic models, as well as sampling and granular-based generative techniques [58]. For the sake of clarity and stability and according to theatrical design, the number of algorithms running in parallel is limited to two or three for the majority of the script. These are complemented in the final section of the piece by real-time lighting colour effects controlled by the incoming AF3-AF4 valence values.

The musical algorithms and the interface to control them were built in Max, under the name *EEGmus*. Figure 5.2 shows the main GUI, with its 4 main components: a preset

[3]In their study, [168] use the F3-F4 pair of sensors. In our experiments we found AF3-AF4 to yield a more responsive output.

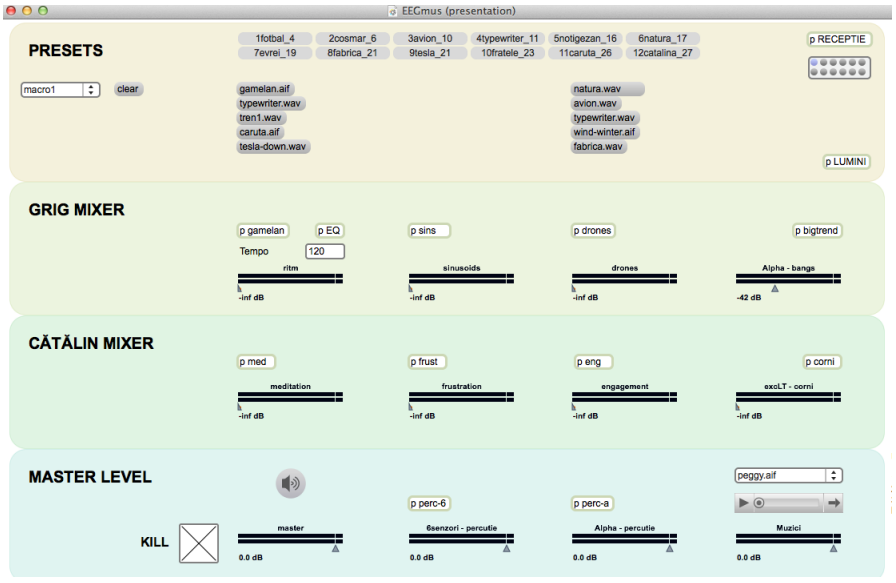


FIGURE 5.2: *EEGmus*, the Max-based suite of algorithms controlled by EEG data.

area, where configurations relating to different parts of the script can be loaded, two mixer areas where the output levels of each algorithm can be set, and a master area where the summed output can be controlled, together with auxiliary effects.

The correspondence between input data and musical algorithm parameters can be defined through a system of macros. These allow the designer to set up four types of control relationships:

- basic proportionality: relative to an input value's trend
- similarity: are two values changing in the same direction?
- opposition: are two values evolving in opposite directions?
- intersection: have two values intersected?

The macro output can be typed either *value* or *bang*, producing a gradual quantification of the chosen relationship (between 0. and 1.) or an atomic trigger when the relationship activates (e.g. when two values start growing in the same direction), respectively. Since we are using two headsets, macros allow us to configure algorithms to use the two subjects' data streams in conjunction – this happens at several times in the piece. Aside from the macro system, several relationships were "hardwired" into the algorithms, thus reducing the superficial complexity of the program.

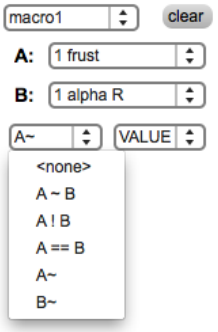


FIGURE 5.3: The macro system inside *EEGmus*, for setting up relationships between two data streams.

5.1.3 Post-performance Analysis

Throughout two series of local and national performances, there emerged a number of recurring musical/EEG highlights between several instances of the piece. Our visual analysis of recorded AF3-AF4 valence data revealed peaks coinciding with intense performative moments or marked alternations of performance style [16]. While our measurements are far less accurate than those produced in controlled conditions, they have shown to be qualitatively dependable, and reflective of the deep structure of the piece, as reproduced in a number of performances.

In a theatre performance environment, the spectator is exposed to a complex set of stimuli, and they constantly, independently select aspects of the performance to focus on. For this reason we chose not to attempt a play-by-play causal analysis, but rather a global look at the peaks and valleys of the entire representation across several subjects. Since the actors were too mobile to allow for a clean EEG signal, we selected the audience subject's EEG for post-performance analysis.

FIGURE 5.4: Actress Cătălina Bălălău being EEG monitored while performing the testimony of witness Notti Gezan, Holocaust survivor.

We used the EDF browser software[4] to produce the AF3-AF4 valence evolution displays shown in Figures 5.5 through 5.7. For a smoother display, a Butterworth low-pass filter with a 0.01Hz cutoff was applied, allowing us to visually observe positive/negative trends and peaks in the signal.

The low temporal resolution and the noisiness of the underlying EEG signal are counterbalanced by the advantage of an extended record, which makes a qualitative analysis possible. We identified several key moments in the piece that were reflected as shifts from positive to negative values, indicating valence changes from the left to the right pre-frontal cortex.

[4] Available at `http://www.teuniz.net/edfbrowser/`.

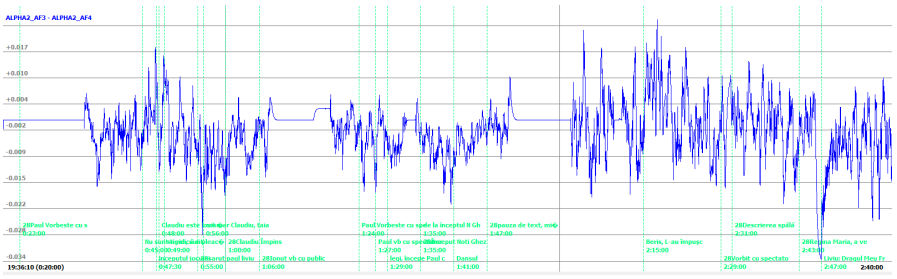


FIGURE 5.5: EEG recording of the spectator headset for the 28 October performance: correlating AF3-AF4 valence peaks (blue) to significant theatrical events (green).

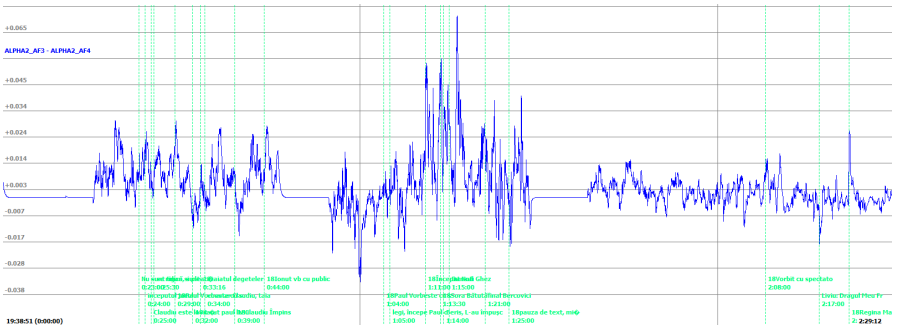


FIGURE 5.6: EEG recording of the spectator headset for the 18 November performance. Each of the three signal sections corresponds to one subject.

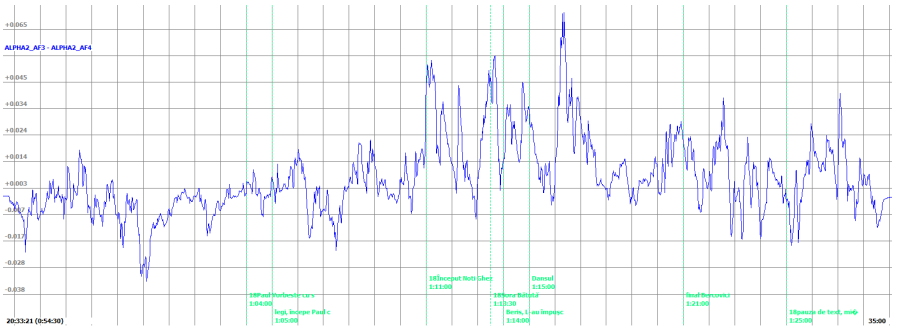


FIGURE 5.7: Zooming in on the 35-minute middle section (18 Nov) reveals alignments between most significant events and AF3-AF4 valence peaks.

One significant moment was the testimony of Holocaust survivor Notti Gezan (see figure 5.4), which consistently produced above average peaks, as is evident in the middle portion of figure 5.7. Interestingly, in the October middle subject (figure 5.5) the polarity seems to be reversed, and we detect negative (left hemisphere) activity instead of positive. Since it is highly unlikely that the subject experienced positive emotions during that monologue (probably the most brutal event recounted in the piece) we are led to conclude that the subject had an opposite valence polarisation, often found in the left-handed [96, 125]. This aspect had limited effects on the sonification, where we harnessed absolute trend evolution and peak amplitudes in parallel to positive-negative valence values.

5.2 Discussion

While data exploration through sonification is a burgeoning field, it is still a long way from displacing the visual medium as the go-to for time series observation. Evidence of this established fact can be seen in the way experiments such as the above are documented: we needed to plot the EEG traces visually, as if we wouldn't simply trust our ears to highlight the essential information for us.

Hermann et al [99] put forward a bold vision for sonification in 50 years, when it might have reached the status of an established and standardised method for display and analysis, as natural as today's mouse and keyboard (we would argue that the video monitor would be a better analogy). We welcome this optimism, and our main insight would be that music and performative art are a significant, probably undervalued [221] path towards this bright future. We approached the potential of musical sound to mediate between dramatic action, performer cognition and audience reception, starting from on a set of superficial brainwave-based emotion models, with encouraging results.

Our observations are consistent with the literature on EEG classification of emotional states by valence and intensity [125, 168]. In the absence of a control sample, it is difficult to ascertain the influence of the sonification on the recorded affective indices. The informal audience feedback indicated an occasional awareness of a correspondence between the soundtrack and the mood of the action, which served the artistic goal of subliminal mediation but does not yet suffice for a stronger scientific conclusion. Our insights are corroborated by similar projects such as Eaton et al's [77], which took place independently and concurrently with our project, where artist-audience mediation is also sonified by means of arousal and valence measurements.

The team plans to continue these explorations in the future, powered by a professional 64-channel EEG system designed for mobile use. Aside from verifying our observations with more robust measurements, the plan is to experiment with other types of algorithms for emotion recognition from real-time EEG analysis, such as mu waves in connection with physical movement [158].

Chapter 6

Conclusions

This thesis is the outcome of three years of research and practice in the area of real-time computer music. We set out to explore different facets of the field and develop new models to address practical issues, advancing the state of the art of music computing. Throughout, we used the construct of dynamic music representations as a guiding frame for the phases of electronic input (machine listening) and output (live electronics, sonification), and their visualisation.

In chapter 1 we set up five specific objectives for this thesis. The first, creating visual models for scored dynamic music processes, was accomplished via our additions to the *AscoGraph* mixed music notation software in chapter 3. The second, developing score-agnostic performance tracking algorithms, was realised and deployed in the form of our rvdtw~ software in chapter 4. Also in chapter 4 we met the goal of comparing audio and score-based followers by way of case studies for rvdtw~ and IRCAM's *Antescofo*, respectively. Chapter 4 ended with our investigation of generative grammars for interactive music score *translation*, addressing our fourth goal. Finally, in chapter 5 we accomplished the last goal through our study and implementation of EEG sonification within an interactive theatre setting, which has been performed across the country.

As with any project of such a broad scope and practical aspirations, there remain countless avenues for further progress. In our introduction, we likened this thesis to a stepping stone—indeed, its shape is more that of a wide step than a tall ladder. In this final chapter, we review our research contributions and look towards possible future developments.

6.1 Summary of Research Results

We now review the main results of the work in this thesis.

Visualisation and notation

In chapter 3 we brought an information visualisation approach to the task of dynamic mixed music notation in *AscoGraph*, the interactive visual interface for the *Antescofo* IMS platform. In section 3.1 we solved the problem of overlapping blocks in the action view, optimising their placement through four novel algorithms derived from bin packing research.

We then turned to the complex issue of representing dynamic durations and processes on the timeline, by proposing visual solutions for the action view (section 3.3.1) and the simulation view (section 3.3.2). We also introduced out-of-time solutions in section 3.4, to display the status of code blocks and the code hierarchy tree outside of the main timeline.

We validated this set of models by applying them to the *Antescofo* unit testing framework, and by rendering richly dynamic mixed music scores.

Tempo modelling

In section 4.1 we introduced rvdtw~ , a Max external object for online audio-to-audio alignment and tempo tracking. Our system is based on a modified online DTW algorithm, which we described in section 4.1.1.

We noted the limitations of using the raw alignment path as a tempo indicator, and in section 4.1.3 implemented four tempo models that help smooth the path in order to better drive an accompaniment, without reference to any symbolic score information. Two of the modelling algorithms are wholly novel: one is based on control theory and the other uses dynamic programming. In section 4.1.4 we the difficulties of evaluating accompaniment performance via established score following methods and propose an alternative framework built on perception-related criteria.

Finally, we dealt with the issue of jumps or variation from the reference by complementing the alignment-based model with a beat tracking component in section 4.1.5. We

developed a switching mechanism that estimates when the alignment path is becoming erratic and enables the beat tracker to take over the tempo computation until the alignment is again reliable. Our practice-led testing showed a qualitative improvement of accompaniment performance for music featuring flexible playing styles over strong beats, as shown in section 4.1.6.

Interactive music applications

All of the work in this thesis has a practical musical end goal in mind. Some of these applications were realised in section 4.2. We built three accompaniment and mixed music systems, each with their particular character and functionality. Due to the flexibility of the Max environment, they can be modified and extended for different situations.

Generative grammars

Our exploration of formal grammars in section 4.3 parallels a rich tradition of computer musicians appealing to formal procedures to automate the generation and organisation of material. We pointed our research towards a yet uncharted goal: the generation of electronic scores from existing acoustic pieces. Our heuristic experiment with score translation indicates the eventual possibility of automating such procedures, at least down to a certain level of abstraction.

Sonification of EEG data

This thesis started with sound and it ends in sound. In chapter 5 we applied a musical representation to raw and processed brainwave data, as part of an interactive theatre piece. We were able to qualitatively observe a validation of EEG emotional state detection literature in the context of audience-actor feedback partly mediated by a dynamic musical narrative.

Software development

All the above work (except for the formal grammars research) took shape in the form of various software. We aimed to write open and reusable code, as exemplified by the online alignment library from section 4.1.2. A full list of the software and media accompanying this thesis is available in appendix A.

6.2 Original Contributions

We summarise the main novel contributions of this thesis:

- Chapter 3: a coherent *visualisation framework* for the notation and display of mixed computer music scores, including optimised placement of electronic action blocks on a timeline and solutions for the representation of dynamic processes, on the timeline and beyond;
- Chapter 4: a robust, score-agnostic, real-time *tempo modelling* engine based on online audio-to-audio alignment;
- Chapter 4: an adaptive *switching mechanism* between the above model and a beat tracker, based on the coupled feedback between the two;
- Chapter 5: a framework for using *sonification in a stage performance* to actively control a soundtrack via the actors' and audience's EEG signals.

6.3 List of Publications

The work herein has been published in a number of journal and conference papers.

Chapter 3 is largely represented in:

- Burloiu, G., Cont, A., and Poncelet, C., forthcoming. A visual framework for dynamic mixed music notation. *Journal of New Music Research (JNMR)* [29];
- Burloiu, G. and Cont, A., 2015. Visualizing Timed, Hierarchical Code Structures in AscoGraph. In *International Conference on Information Visualisation* [26];
- Burloiu, G. and Cont, A., 2015. Non-overlapping, Time-coherent Visualisation of Action Commands in the AscoGraph Interactive Music User Interface. In *International Conference on Technologies for Music Notation and Representation (TENOR)* [27];

Chapter 4 contains material from:

- Burloiu, G., 2016. An Online Tempo Tracker for Automatic Accompaniment based on Audio-to-audio Alignment and Beat Tracking. In *Sound and Music Computing conference (SMC)* [23];

- Burloiu, G., 2016. Online Score-agnostic Tempo Models for Automatic Accompaniment. In *International Workshop on Machine Learning and Music (MML)* [24];
- Burloiu, G., forthcoming. A Robust Online DTW-based Audio Alignment Tool for Max/MSP. *Polytechnical University of Bucharest. Scientific Bulletin* [25];
- Burloiu, G., 2014. An online audio alignment tool for live musical performance. In *Electronics and Telecommunications (ISETC)* [22].

Chapter 5 builds on work documented in:

- Burloiu, G., Berceanu, A., and Creţu, C., 2016. EEG-powered Soundtrack for Interactive Theatre. *In Workshop on Auditory Neuroscience, Cognition and Modelling (WANCM)* [28];
- Creţu, C., Berceanu, A., and Burloiu, G., 2015. EEG-based Interactive Stage Music. *ICT in Musical Field*, VI(2):83—91 [58];
- Berceanu, A., Creţu, C., Burloiu, G., and Cirneci, D., 2015. Extension of Performativity by a BCI. *Studia UBB Dramatica*, LX(2):77—100 [16];

6.4 Future Perspectives

While our research has resulted in self-contained, ready-to-use software, there remains a multitude of work to be done, both in the modelling and development areas. We first describe potential work related to our two major software projects, before turning to more general research directions.

6.4.1 *AscoGraph*

As we noted in chapter 3, a significant portion of the work presented is still in prototype or mock-up form. While our design has aimed to anticipate real work conditions, it is inevitable that the remaining implementation will turn up additional problems. We hope to have provided a solid roadmap for how to approach future issues, but acknowledge that the greater problem of representing dynamic processes remains to be fully solved.

In addition, we would point to further refinement of the action view model. At wide zoom levels, block aggregation of neighbouring groups might be implemented, instead of drawing them separately. Semantic zooming might also be of help: in the broad view

it is pointless to burden the scene with textual detail. Support for filters and tracks also leaves room for improvement, perhaps drawing on the concepts from the simulation view.

As the *Antescofo* reactive language evolves, *AscoGraph* should keep adapting its visual models to new abilities – like for instance, *patterns*, which help specify complex logical conditions for use in a whenever, or *objects*, which extend the language with aspects of object-oriented programming. Also, similarly to existing coding environments, we bring up the development of debugger-oriented features to further facilitate the testing of *Antescofo* programs.

Finally, the system could be further opened up to user customization through a powerful API and a robust import/export system that empowers musicians to integrate the visual notation into their personal authoring and performance workflows.

6.4.2 *rvdtw˜*

The rvdtw˜ object is currently at version 0.3. There is a lot of work left until v1.0, and we intend to keep improving the software, especially if it gains community adoption.[1]

As we did for the oDTW mechanism in section 4.1.2, we intend to abstract the tempo models into a separate C++ library for public use and evaluation, together with oDTW or in combination with external frameworks.

Regarding evaluation, a perceptually grounded framework based on a cross-genre database of backing tracks and isolated performances is needed in order to ensure measurable progress.[2] Our proposed embedding of beat tracking into the alignment process is certainly not the only possible method, and so far we have relied on piecewise experimentation to move forward—we need a reliable evaluation framework in order to make systematic progress. We also see potential in neurocognitive approaches to synchronicity tracking. Real-time neuroimaging techniques might be able to make such an evaluation model possible, where we bypass the step of the subject actively pointing out timespans

[1] Although the basic alignment object has been public since late 2014, we had not advertised it due to the lack of tempo modelling. With the updates completed in summer 2016, we have started sharing the current experimental version. The initial Facebook post on the Max community group has garnered some promising interest: `https://www.facebook.com/groups/maxmspjitter/10155145111774392/` .

[2] The MIR community has long been aware of the necessity for subjectively relevant assessment criteria for music performance modelling (see e.g. [93]). The MIREX evaluation framework has been making significant progress towards that goal.

of correct synchronisation, and instead interpret this information from EEG data. Basic research suggests that event-related potential (ERP) data might be used to measure adaptation and synchronisation with coupled sound stimuli [3].

Moreover, we are looking to develop new tempo models that make integral use of both performance alignment and beat tracking. The alteration of the Δt value described in section 4.1.5 is a start, but we might conceive models from the ground up with this configuration in mind.

Conversely, we might consider bringing the beat tracker up to equal footing with the alignment, rather than simply acting as a backup. This would involve updating the beat tracker with solid tempo information from the aligner. Still, we must be careful to keep the two models sufficiently orthogonal, so as to benefit from switching between them.

In view of the practical use of our system, we return to the question of the adequacy of a single master tempo curve to describe the temporal dynamics of a musical accompaniment. This brings up artistic considerations as well as technical ones. Among the research directions worth considering are multi-agent following [7] and asynchrony compensation [217] to capture the timing nuances of several musical facets.

Perhaps the key is not to be found in a single accompaniment track, but in a database of performances of the same piece at slightly different dynamics, which can be concatenated in real time [169]. Further on, from a sufficiently large database mid-level representations could be extracted that capture the essence of certain tempo dynamics (deep learning might be appropriate), and can be altered to create variations and enrich the accompaniment pool. Coupled with multi-agent following, this could result in a many-to-many performance tracker with high robustness and expressivity.

Finally from a creative perspective, we look towards ways of harnessing the eccentricities of our existing tools to produce artificial accompaniments that, while "unnatural" to some extent, can prove musically fruitful.

6.4.3 General Directions

While the conception of an all-encompassing theoretical framework has been outside of our scope, we do hope that our dynamic music representation guidelines might point

towards a theory of discovery [190] of sorts. In chapter 1 we stated our overall aim to contribute to an interactive music ecosystem. We now conclude this text by putting forward some perspectives on how this ecosystem might be further populated.

One possible direction is the combination of performance following and sonification. We can see rudimentary examples of this in e.g. instrumental tuition software where sonic (and visual) cues are prompted by certain (mis)alignment events. We see a lot of potential, both in sonifying alignment data (relating to anticipation, following error, tempo model switching etc), and in modulating data sonification via score following dynamics.

Related to Dannenberg's HCMP model [62], we are also looking to extend our listening and reactive machines to different input sources, such as motion tracking and video. Beyond basic sonification and gesture classification, these types of data can be used to transmit intention and synchronisation information between human and machine that is otherwise absent from the audio stream. There exist already examples of *Antescofo*-based multi-modal following frameworks [174] which include gesture following. DTW has also been used for open-ended, potentially incomplete gesture classification [209], a task which might be also achievable by rvdtw~ .

We also intend to expand our investigations into automating interactive scenario generation. As our experience in section 4.3 has shown, there is potential in applying automatic composition strategies to the mixed music realm.

Last but not least, we intend to engage in more practice-led research [133], producing interactive audio work that reaps the benefits of our technology and informs us on how to develop it. We have seen instances of this workflow in the case studies from sections 3.5, 4.1.6 and 4.2. We plan to build and pursue a systematic practice that contributes to bringing humans and computers into more coherent, immediate and ultimately emotive arrangements.

Appendix A

Supporting Audio/Video Media

The following is a list of materials included on the CD accompanying this text. Most of them are also publicly available in online repositories. For assistance with obtaining or deploying any of the files, please email gburloiu@gmail.com .

- /music/case/ : *Case mici* Max/Antescofo patches and live video recording
- /music/interfata/ : *INTER@FAȚA* Max patches and media
- /music/triangolo/ : *Triangolo Aperto* Max/Antescofo patches and audio recordings
- /software/eegOSC/ : eegOSC for OSX & Emotiv EPOC headsets
- /software/RVdtw/ : rvdtw~ Max Package
- /src/eegOSC/ : eegOSC source code
- /src/RVdtw/ : rvdtw~ source code
- /src/RVdtw/oDTW/ : oDTW C++ library

Appendix B

rvdtw~ Object Reference

Track position and tempo relative to a pre-recorded reference take.[1]

B.1 Description

Use the rvdtw~ object for automatic accompaniment or sequence matching applications where you need to follow a stored reference. The **reference** is encoded as a sequence of numeric vectors, which can be entered directly or extracted from an audio signal. Likewise, the real-time **target** can be input as a signal or as sequential feature vectors.

B.2 Arguments

- score-name [symbol] *OPTIONAL*

The name of a text file or a buffer~ object used to preload the reference sequence.

B.3 Messages

- input

The word input, followed by a number, selects the object's state. `0` is *OFF*; `1` is *SCORE* (to read the reference); `2` is *LIVE* (to process the target).

[1] The presentation of this appendix follows the Max Reference style; see e.g. `https://docs.cycling74.com/max7/maxobject/cycle~`. Specifications herein are valid as of version 0.31 of rvdtw~ .

- score_size

The word score_size, followed by a number, specifies the number of vectors expected for the reference, and allocates the corresponding memory.

- feats

The word feats, followed by a list of numbers, processes a new feature vector.

- *signal*

The object processes signals to produce (via chromagram extraction) sequences of feature vectors, processed similarly to the *feats* input.

- read

The word read, followed by a symbol, points to a text file to load as a reference vector sequence. If the file is missing, the object looks for a buffer~ object with the same name, to be processed as a signal. Any information from *score_size* is discarded; the reference size is determined by the loaded file or buffer length.

- readacco

The word readacco, followed by a symbol, points to the buffer~ object to be used as an accompaniment track. This input is optional and only serves to provide additional beat information to the tracking engine.

- write

Depending on the *input* state, the word write prompts a save dialog to export (1) the reference sequence as a text file, or (2) the marker alignment times and miscellaneous test information as a CSV file.

- start

Sets the input mode to *LIVE* and the current time to zero.

- stop

Sets the input mode to *OFF* and the current time to zero.

- follow

The word follow, followed by `0` or `1`, turns tracking on and off. When *follow* is off, the relative tempo is maintained at the current value (initially 1), and the alignment position advances accordingly.

- sensitivity

The word follow, followed by a float number, sets how easily the tempo tracking is activated. The initial value is `1`, meaning the tracker is always active. The smaller the value, the more the target speed must differ from the reference in order to engage the tracker. A zero value is equivalent to *follow 0*.

- elasticity

The word elasticity, followed by a float number, sets how reactive the tracker is to changes in target speed. The initial value is `1`. A zero value is equivalent to *follow 0*. Values larger than `1` produce exaggerated, potentially unstable relative tempos.

B.4 Outlets

The `rvdtw˜` object has five outlets. From left to right, they are:

- t [int]

Outputs the current elapsed time frame t.

- h [int]

Outputs the current alignment index frame h_t.

- misc [list]

Outputs one of several lists:
(a) h_real [float] (accompaniment index $\tilde{h}_t$) [float] (scaled acc. index $\frac{\tilde{h}_t}{n}$) [int] (tempo mode: `0` for *OFF*, `1` for *ALIGN*, `2` for *BEAT*);
(b) marker [int] (number of marker just reached);
(c) b_err [int] (the backwards DTW deviation δ_t^{BACK}) [float] (the local tempo τ^{FW} smoothed over a number of recent frames);

(d) beat_err [int] (the average of the past 3 differences between reference and accompaniment beats, in number of frames).

- beat [bang]

Outputs every time a beat is detected in the target signal.

- tempo [float]

Outputs the current relative tempo τ_t.

Appendix C

oDTW C++ Class Reference

Class for performing online DTW on sequences of feature vector frames, with optional backwards DTW phase for path adjustment.

`#include <oDTW.h>`

C.1 Public Member Functions

oDTW(int windowSize_ = FSIZE, int backWindowSize_ = BSIZE, bool backActive_ = true, unsigned int params_ = 12)

Constructor. Parameters are the online DTW window size (in number of frames), the backward DTW window size, a flag to activate the backward DTW, and the expected number of features in an input vector.

~oDTW()

Destructor.

void setParams(int params_) Set number of features in an input frame.

unsigned int setScoreSize(long v)

Set score (reference) size (in number of frames) and clear the corresponding memory. Returns number of parameters if succeeded, 0 if incorrect length v.

`unsigned int processScoreFV(double *tfeat)`

Add a score (reference) feature vector. Returns the location number of the current vector.

`void processLiveFV(double *tfeat)`

Add a live (target) feature vector.

`void addMarkerToScore(unsigned int frame = 0)`

Add a marker to the score reference at location `frame`. If the parameter is zero, the marker is added at the current location.

`void addMarkerToLive(unsigned int frame = 0)`

Add a marker to the live target at location `frame`. If the parameter is zero, the marker is added at the current location.

`unsigned int getMarkerFrame(long here)`

Get location number for a certain (score) marker.

`unsigned int getMarkerCount()`

Get total marker count.

`void start()`

Reset the internal state (except for the score) and point back to the start.

`unsigned int getT()`

Get the current live location t.

`unsigned int getH()`

Get the current reference index h_t.

`void setH(unsigned int to_h)`

Set new reference index. Use with caution—causes the system to effectively jump in time!

`unsigned int getHistory(unsigned int from_t)`

Get historic h_t value.

`unsigned int getFsize()`

Get online DTW window size, in number of frames.

vector<vector<double> > getBackPath()

Get backwards path data: error δ_t^{BACK}, index t, index h_t, local tempo τ^{FW} for the past backWindowSize_ frames.

bool isRunning()

Returns true if the end has not been reached and live vectors can still be processed, and false otherwise.

bool isScoreLoaded()

Returns true if all score features have been entered, and false otherwise.

C.2 Example Usage

(1) Include the library header.

```
#include "oDTW.h"
```

(2) Instantiate the oDTW object.

```
oDTW *dtw = new oDTW();
// OR: oDTW *dtw = new oDTW(128, 512, false, 12);
```

(3) Set the score size. This is the number of frames the object will allocate memory for.

```
dtw->setScoreSize(ysize);
```

(4) In a loop, fill a feature vector with values and then add them to the score.

```
double f_feat[12];
// ... fill vector
dtw->processScoreFV(f_feat);
```

When you're done, you can check whether the score is fully loaded:

```
if (dtw->isScoreLoaded()) {
  dtw->start(); // start "listening" for live features
  //...
}
```

(5) In the "live" mode, you similarly fill a feature vector and process it frame by frame:

```
dtw->processLiveFV(f_feat);
```

At any point, you can check whether the end has been reached, and do something with the current t and/or h_t coordinates:

```
if (dtw->isRunning()) {
  int t = dtw->getT();
  int h = dtw->getH();
  // do something
}
```

When you're done, you can start() a new run.

Bibliography

[1] Ackoff, R. L., 1989. From data to wisdom. *Journal of applied systems analysis*, 16(1):3–9.

[2] Agostini, A. and Ghisi, D., 2012. Bach: An Environment for Computer-Aided Composition in Max. In *International Computer Music Conference*. Ljubljana, Slovenia.

[3] Allefeld, C., Frisch, S., and Schlesewsky, M., 2005. Detection of early cognitive processing by event-related phase synchronization analysis. *Neuroreport*, 16(1):13–16.

[4] Allombert, A., Desainte-Catherine, M., and Assayag, G., 2008. Iscore: A System for Writing Interaction. In *International Conference on Digital Interactive Media in Entertainment and Arts*, DIMEA '08, pp. 360–367. ACM, New York, NY, USA.

[5] Arias, J., Desainte-Catherine, M., Salvati, S., and Rueda, C., 2014. Executing Hierarchical Interactive Scores in ReactiveML. In *Journées d'Informatique Musicale*. Bourges, France.

[6] Arzt, A. and Widmer, G., 2010. Simple tempo models for real-time music tracking. In *Sound and Music Computing conference (SMC)*.

[7] Arzt, A. and Widmer, G., 2015. Real-time music tracking using multiple performances as a reference. In *International Society for Music Information Retrieval conference (ISMIR)*.

[8] Arzt, A., Widmer, G., and Dixon, S., 2008. Automatic Page Turning for Musicians via Real-Time Machine Listening. In *European Conference on Artificial Intelligence (ECAI)*, pp. 241–245.

[9] Arzt, A., Widmer, G., and Dixon, S., 2012. Adaptive distance normalization for real-time music tracking. In *Signal Processing Conference (EUSIPCO), 2012 Proceedings of the 20th European*, pp. 2689–2693. IEEE.

[10] Arzt, A., Frostel, H., Gadermaier, T., Gasser, M., Grachten, M., and Widmer, G., 2015. Artificial intelligence in the concertgebouw. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI), Buenos Aires, Argentina*.

[11] Assayag, G., Rueda, C., Laurson, M., Agon, C., and Delerue, O., 1999. Computer Assisted Composition at Ircam: From PatchWork to OpenMusic. *Computer Music Journal*, 23(3).

[12] Bakker, R. S., 2014. Discontinuity Thesis: A 'Birds of a Feather' Argument Against Intentionalism. URL `https://rsbakker.wordpress.com/2014/06/16/discontinuity-thesis-a-birds-of-a-feather-argument-against-intentionalism/`. [Online; accessed 13-August-2016].

[13] Beer, R. D., 2000. Dynamical approaches to cognitive science. *Trends in cognitive sciences*, 4(3):91–99.

[14] Bel, B. and Kippen, J., 1992. Bol Processor Grammars. In M. Balaban, ed., *Understanding Music with AI*, pp. 366–400. AAAI Press.

[15] Beltrami, E., 2014. *Mathematics for dynamic modeling*. Academic press.

[16] Berceanu, A., Creţu, C., Burloiu, G., and Cirneci, D., 2015. Extension of Performativity by a BCI. *Studia UBB Dramatica*, LX(2):77–100.

[17] Bernardini, N. and Vidolin, A., 2005. Sustainable live electro-acoustic music. In *Proceedings of the International Sound and Music Computing Conference*.

[18] Berry, G. and Cosserat, L., 1984. The ESTEREL synchronous programming language and its mathematical semantics. In *International Conference on Concurrency*, pp. 389–448. Springer.

[19] Bird, R. and Wadler, P., 1988. *Introduction to functional programming*, vol. 1. Prentice Hall New York.

[20] Brouse, A., 2004. A young person's guide to brainwave music. *HorizonZero: Digital Art+ Culture*.

[21] Brown, J. C. and Puckette, M. S., 1992. An efficient algorithm for the calculation of a constant Q transform. *The Journal of the Acoustical Society of America*, 92(5):2698–2701.

[22] Burloiu, G., 2014. An online audio alignment tool for live musical performance. In *Electronics and Telecommunications (ISETC)*.

[23] Burloiu, G., 2016. An Online Tempo Tracker for Automatic Accompaniment based on Audio-to-audio Alignment and Beat Tracking. In *Sound and Music Computing conference (SMC)*.

[24] Burloiu, G., 2016. Online Score-agnostic Tempo Models for Automatic Accompaniment. In *International Workshop on Machine Learning and Music (MML)*.

[25] Burloiu, G., forthcoming. A Robust Online DTW-based Audio Alignment Tool for Max/MSP. *Polytechnical University of Bucharest. Scientific Bulletin.*

[26] Burloiu, G. and Cont, A., 2015. Visualizing Timed, Hierarchical Code Structures in AscoGraph. In *International Conference on Information Visualisation*. University of Barcelona, Barcelona, Spain.

[27] Burloiu, G. and Cont, A., 2015. Non-overlapping, Time-coherent Visualisation of Action Commands in the AscoGraph Interactive Music User Interface. In *International Conference on Technologies for Music Notation and Representation (TENOR)*.

[28] Burloiu, G., Berceanu, A., and Crețu, C., 2016. EEG-powered Soundtrack for Interactive Theatre. In *Workshop on Auditory Neuroscience, Cognition and Modelling (WANCM)*.

[29] Burloiu, G., Cont, A., and Poncelet, C., forthcoming. A visual framework for dynamic mixed music notation. *Journal of New Music Research (JNMR)*.

[30] Buxton, W., Patel, S., Reeves, W., and Baecker, R., 1979. The evolution of the SSSP score-editing tools. *Computer Music Journal*, 3(4):14–25.

[31] de Campo, A., 2007. Toward a data sonification design space map.

[32] Cano, P., Loscos, A., and Bonada, J., 1999. Score-performance matching using HMMs. In *Proceedings of the International Computer Music Conference*.

[33] Carabias-Orti, J., Rodriguez-Serrano, F., Vera-Candeas, P., Ruiz-Reyes, N., and Canadas-Quesada, F., 2015. An audio to score alignment framework using spectral factorization and dynamic time warping. In *16th International Society for music information retrieval conference*, pp. 742–748.

[34] Card, S. K., Mackinlay, J. D., and Shneiderman, B., 1999. *Readings in information visualization: using vision to think*. Morgan Kaufmann.

[35] Celerier, J.-M., Baltazar, P., Bossut, C., Vuaille, N., Couturier, J.-M., and Desainte-Catherine, M., 2015. OSSIA: towards a unified interface for scoring time and interaction. In M. Battier, J. Bresson, P. Couprie, C. Davy-Rigaux, D. Fober, Y. Geslin, H. Genevois, F. Picard, and A. Tacaille, eds., *International Conference on Technologies for Music Notation and Representation*, pp. 81–90. Institut de Recherche en Musicologie, Paris, France.

[36] Cemgil, A. T., Kappen, B., Desain, P., and Honing, H., 2000. On tempo tracking: Tempogram representation and Kalman filtering. *Journal of New Music Research*, 29(4):259–273.

[37] Chadabe, J., 1984. Interactive composing: An overview. *Computer Music Journal*, 8(1):22–27.

[38] Chemero, A., 2011. *Radical embodied cognitive science*. MIT press.

[39] Chion, M., 1983. *Guide des objets sonores: Pierre Schaeffer et la recherche musicale*. Bibliothèque de recherche musicale. Buchet/Chastel.

[40] Chomsky, N., 1957. *Syntactic Structures*. Mouton, The Hague.

[41] Clark, A., 2001. Unsupervised induction of stochastic context-free grammars using distributional clustering. In *Proceedings of the 2001 workshop on Computational Natural Language Learning-Volume 7*, p. 13. Association for Computational Linguistics.

[42] Clay, A. and Freeman, J., 2010. Preface: Virtual Scores and Real-Time Playing. *Contemporary Music Review*, 29(1):1–1.

[43] Clayton, M., Sager, R., and Will, U., 2005. In time with the music: the concept of entrainment and its significance for ethnomusicology. In *European meetings in ethnomusicology.*, vol. 11, pp. 1–82. Romanian Society for Ethnomusicology.

[44] Coduys, T. and Ferry, G., 2004. Iannix Aesthetical/Symbolic Visualisations for Hypermedia Composition. In *Sound and Music Computing*.

[45] Coffman, J., E. G., Garey, M. R., Johnson, D. S., and Tarjan, R. E., 1980. Performance Bounds for Level-Oriented Two-Dimensional Packing Algorithms. *SIAM J. Comput.*, (9):808–826.

[46] Coffy, T., Giavitto, J.-L., and Cont, A., 2014. AscoGraph: A User Interface for Sequencing and Score Following for Interactive Music. In *International Computer Music Conference*. Athens, Greece.

[47] Collins, K., Kapralos, B., and Tessler, H., 2014. *The Oxford Handbook of Interactive Audio*. Oxford Handbooks. Oxford University Press.

[48] Collins, N., McLean, A., Rohrhuber, J., and Ward, A., 2003. Live coding in laptop performance. *Organised sound*, 8(3):321–330.

[49] Cont, A., 2008. ANTESCOFO: Anticipatory Synchronization and Control of Interactive Parameters in Computer Music. In *International Computer Music Conference*. Belfast.

[50] Cont, A., 2010. A coupled duration-focused architecture for realtime music to score alignment. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 32(6):974–987.

[51] Cont, A., 2011. On the creative use of score following and its impact on research. In *Sound and Music Computing conference (SMC)*.

[52] Cont, A., 2013. *Real-time Programming and Processing of Music Signals*. Habilitation à diriger des recherches, Université Pierre et Marie Curie - Paris VI.

[53] Cont, A., Schwarz, D., Schnell, N., and Raphael, C., 2007. Evaluation of Real-Time Audio-to-Score Alignment. In *International Symposium on Music Information Retrieval (ISMIR)*. Vienna, Austria.

[54] Cont, A., Echeveste, J., Giavitto, J.-L., and Jacquemard, F., 2012. Correct Automatic Accompaniment Despite Machine Listening or Human Errors in Antescofo. In *International Computer Music Conference*. IRZU - the Institute for Sonic Arts Research, Ljubljana, Slovenia.

[55] Cont, A., Echeveste, J., and Giavitto, J.-L., 2014. The cyber-physical system approach for automatic music accompaniment in Antescofo. *The Journal of the Acoustical Society of America*, 135(4):2377–2377.

[56] Cook, P., 2001. Principles for designing computer music controllers. In *Proceedings of the 2001 conference on New interfaces for musical expression*, pp. 1–4. National University of Singapore.

[57] Cope, D., 1996. *Experiments in Musical Intelligence*. A-R Editions, Madison, WI.

[58] Crețu, C., Berceanu, A., and Burloiu, G., 2015. EEG-based Interactive Stage Music. *ICT in Musical Field*, VI(2):83–91.

[59] Cycling '74, 2016. Max is a visual programming language for media. URL `https://cycling74.com/products/max/`.

[60] D. Fober, S. L., Y. Orlarey, 2012. INScore - An Environment for the Design of Live Music Scores. In *Proceedings of the Linux Audio Conference - LAC*.

[61] Dannenberg, R. B., 1984. An on-line algorithm for real-time accompaniment. In *International Computer Music Conference (ICMC)*.

[62] Dannenberg, R. B., 2012. Human computer music performance. *Dagstuhl Follow-Ups*, 3.

[63] Dannenberg, R. B. and Hu, N., 2003. Polyphonic audio matching for score following and intelligent audio editors. *Computer Science Department*, p. 507.

[64] Davies, H., 1998. Electronic instruments. *Grove music online*.

[65] Davis, S. B. and Mermelstein, P., 1980. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 28(4):357–366.

[66] Dean, R. T., 2009. *The Oxford handbook of computer music*. Oxford University Press.

[67] Desainte-Catherine, M. and Allombert, A., 2005. Interactive scores : A model for specifying temporal relations between interactive and static events. *Journal of New Music Research*, 34:361–374.

[68] Di Scipio, A., 2003. 'Sound is the interface': from interactive to ecosystemic signal processing. *Organised Sound*, 8(03):269–277.

[69] Dixon, S., 2005. Live tracking of musical performances using on-line time warping. In *International Conference on Digital Audio Effects (DAFx)*, pp. 92–97.

[70] Dixon, S. and Widmer, G., 2005. MATCH: A Music Alignment Tool Chest. In *ISMIR*, pp. 492–497.

[71] Dixon, S., Goebl, W., and Cambouropoulos, E., 2006. Perceptual smoothness of tempo in expressively performed music. *Music Perception: An Interdisciplinary Journal*, 23(3):195–214.

[72] Drummond, J., 2009. Understanding interactive systems. *Organised Sound*, 14(02):124–133.

[73] Duan, Z. and Pardo, B., 2011. A state space model for online polyphonic audio-score alignment. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pp. 197–200. IEEE.

[74] Duan, Z. and Pardo, B., 2011. Soundprism: An Online System for Score-Informed Source Separation of Music Audio. *IEEE Journal of Selected Topics in Signal Processing*, 5(6):1205–1215.

[75] Durkin, A., 2014. *Decomposition: A Music Manifesto*. Pantheon.

[76] Duvinage, M., Castermans, T., Petieau, M., Hoellinger, T., Cheron, G., and Dutoit, T., 2013. Performance of the Emotiv Epoc headset for P300-based applications. *BioMedical Engineering OnLine*, 12(1):1–15.

[77] Eaton, J., Jin, W., and Miranda, E., 2014. The Space Between Us. A Live Performance with Musical Score Generated via Emotional Levels Measured in EEG of One Performer and an Audience Member. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pp. 593–596. Goldsmiths, University of London, London, United Kingdom.

[78] Echeveste, J., Cont, A., Giavitto, J.-L., and Jacquemard, F., 2013. Operational semantics of a domain specific language for real time musician-computer interaction. *Discrete Event Dynamic Systems*, 23(4):343–383.

[79] Echeveste, J., Giavitto, J.-L., and Cont, A., 2013. A Dynamic Timed-Language for Computer-Human Musical Interaction. Research Report RR-8422, INRIA.

[80] Echeveste, J., Giavitto, J.-L., and Cont, A., 2015. Programming with Events and Durations in Multiple Times: The Antescofo DSL. *ACM Transactions on Programming Languages and Systems*. (submitted).

[81] Eimert, H., Stockhausen, K., et al., 1958. Electronic music. Tech. rep., T. Presser Company.

[82] Ellis, D. P. W. and Rosenthal, D. F., 1995. *Mid-level representations for computational auditory scene analysis*. Perceptual Computing Section, Media Laboratory, Massachusetts Institute of Technology.

[83] Ewert, S. and Müller, M., 2008. Refinement strategies for music synchronization. In *International Symposium on Computer Music Modeling and Retrieval*, pp. 147–165. Springer.

[84] Fineberg, J., 2000. Guide to the basic concepts and techniques of spectral music. *Contemporary Music Review*, 19(2):81–113.

[85] Flowers, J. H., 2005. Thirteen years of reflection on auditory graphing: Promises, pitfalls, and potential new directions. *Faculty Publications, Department of Psychology*, p. 430.

[86] Fodor, J. A., 1983. *The modularity of mind: An essay on faculty psychology*. MIT press.

[87] Franklin, G. F., Powell, J. D., and Workman, M. L., 1998. *Digital control of dynamic systems*, vol. 3. Menlo Park: Addison-Wesley.

[88] Freeman, J., 2010. Web-based collaboration, live musical performance and open-form scores. *International Journal of Performance Arts and Digital Media*, 6(2):149–170.

[89] Fremerey, C., Müller, M., and Clausen, M., 2010. Handling Repeats and Jumps in Score-performance Synchronization. In *International Society for Music Information Retrieval conference (ISMIR)*, pp. 243–248.

[90] Fujishima, T., 1999. Realtime chord recognition of musical sound: A system using common lisp music. In *Proc. ICMC*, vol. 1999, pp. 464–467.

[91] Giavitto, J.-L., Cont, A., Echeveste, J., and Members, M. T., 2015. *Antescofo: A Not-so-short Introduction to Version 0.x.* MuTant Team-Project, IRCAM, Paris, France. URL `http://support.ircam.fr/docs/Antescofo/AntescofoReference.pdf`.

[92] Gibson, J. J., 1979. *The ecological approach to visual perception: classic edition.* Psychology Press.

[93] Goebl, W., Dixon, S., De Poli, G., Friberg, A., Bresin, R., and Widmer, G., 2008. Sense in expressive music performance: Data acquisition, computational studies, and models. *Sound to sense-sense to sound: A state of the art in sound and music computing*, pp. 195–242.

[94] Grachten, M., Gasser, M., Arzt, A., and Widmer, G., 2013. Automatic alignment of music performances with structural differences. In *International Society for Music Information Retrieval conference (ISMIR).*

[95] Grubb, L. and Dannenberg, R. B., 1998. Enhanced vocal performance tracking using multiple information sources. In *Proceedings of the International Computer Music Conference*. Internation Computer Music Association.

[96] Hamann, S. and Canli, T., 2004. Individual differences in emotion processing. *Current opinion in neurobiology*, 14(2):233–238.

[97] Harren, R. and van Stee, R., 2009. Improved absolute approximation ratios for two-dimensional packing problems. In *International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX), 177–189.*

[98] Harris, M., Smaill, A., and Wiggins, G., 1991. *Representing music symbolically.* University of Edinburgh, Department of Artificial Intelligence.

[99] Hermann, T., Hunt, A., and Neuhoff, J. G., 2011. Introduction. In T. Hermann, A. Hunt, and J. G. Neuhoff, eds., *The Sonification Handbook*, chap. 1, pp. 1–6. Logos Publishing House, Berlin, Germany. URL `http://sonification.de/handbook/chapters/chapter1/`.

[100] Hermann, T., Hunt, A., and Neuhoff, J. G., eds., 2011. *The Sonification Handbook*. Logos Publishing House, Berlin, Germany. URL `http://sonification.de/handbook`.

[101] Hils, D. D., 1992. Visual languages and computing survey: Data flow visual programming languages. *Journal of Visual Languages & Computing*, 3(1):69–101.

[102] Hinterberger, T. and Baier, G., 2005. Poser: Parametric orchestral sonification of eeg in real-time for the self-regulation of brain states. *IEEE Trans. Multimedia*, 12:70.

[103] Holmes, T., 2012. *Electronic and experimental music: technology, music, and culture*. Routledge.

[104] ten Holt, G. A., Reinders, M. J., and Hendriks, E., 2007. Multi-dimensional dynamic time warping for gesture recognition. In *Thirteenth annual conference of the Advanced School for Computing and Imaging*, vol. 300.

[105] Holtzman, S., 1981. Using generative grammars for music composition. *Computer Music Journal*, 5(1):51–64.

[106] Honing, H., 2001. From time to time: The representation of timing and tempo. *Computer Music Journal*, 25(3):50–61.

[107] Honing, H., 2005. Timing is tempo-specific. In *Proceedings of the International Computer Music Conference*, pp. 359–362.

[108] Hu, N., Dannenberg, R. B., and Tzanetakis, G., 2003. Polyphonic audio matching and alignment for music retrieval. *Computer Science Department*, p. 521.

[109] Huron, D. B., 2006. *Sweet anticipation: Music and the psychology of expectation*. MIT press.

[110] Johnston, W. M., Hanna, J., and Millar, R. J., 2004. Advances in dataflow programming languages. *ACM Computing Surveys (CSUR)*, 36(1):1–34.

[111] Jordà, S., 2005. *Digital Lutherie: Crafting musical computers for new musics' performance and improvisation*. Ph.D. thesis, Universitat Pompeu Fabra.

[112] Kaipainen, M. and Thur, C., 2015. System and method for providing exercise in playing a music instrument. URL `https://www.google.com/patents/US9218748`. US Patent 9,218,748.

[113] Kemp, B. and Olivan, J., 2003. European data format 'plus'(EDF+), an EDF alike standard format for the exchange of physiological data. *Clinical Neurophysiology*, 114(9):1755–1761.

[114] Knuth, D., 2009. *The Art of Computer Programming*. Art of Computer Programming. Prentice Hall. URL `https://books.google.ro/books?id=5X3nXwAACAAJ`.

[115] Kramer, G., 1994. Some organizing principles for representing data with sound. In *Auditory Display: Sonification, Audification and Auditory Interface*, vol. 18, pp. 185–221. Addison-Wesley.

[116] Kramer, G., Walker, B., Bonebright, T., Cook, P., Flowers, J. H., Miner, N., and Neuhoff, J., 1999. Sonification report: Status of the field and research agenda. *International Conference for Auditory Display (ICAD)*.

[117] Kurtz, M., 1992. *Stockhausen : a biography*. Faber and Faber.

[118] Large, E. W., 1995. Beat tracking with a nonlinear oscillator. In *Working Notes of the IJCAI-95 Workshop on Artificial Intelligence and Music*, vol. 24031.

[119] Lavi, Y. and Begelfor, E., 2014. Automatic score following. URL `https://www.google.com/patents/US8660678`. US Patent 8,660,678.

[120] Leman, M., 2008. *Embodied music cognition and mediation technology*. Mit Press.

[121] Lerch, A., 2012. *An introduction to audio content analysis: Applications in signal processing and music informatics*. John Wiley & Sons.

[122] Lervig, M. C., 2003. From music to 3D scenography and back again. In *Production methods*, pp. 60–76. Springer.

[123] Leslie, G. and Mullen, T., 2011. MoodMixer: EEG-based Collaborative Sonification. In *NIME*, pp. 296–299. Citeseer.

[124] Lexer, S., 2012. *Live Electronics in Live Performance: A Performance Practice Emerging from the piano+ used in Free Improvisation.* Ph.D. thesis, Goldsmiths, University of London.

[125] Lin, Y.-P., Wang, C.-H., Wu, T.-L., Jeng, S.-K., and Chen, J.-H., 2009. EEG-based emotion recognition in music listening: A comparison of schemes for multiclass support vector machine. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 489–492. IEEE.

[126] Lyon, E., 2012. *Designing audio objects for Max/MSP and Pd*. AR Editions, Inc.

[127] Mackinlay, J., 1986. Automating the design of graphical presentations of relational information. *Acm Transactions On Graphics (Tog)*, 5(2):110–141.

[128] Macrae, R. and Dixon, S., 2010. Accurate Real-time Windowed Time Warping. In *International Society for Music Information Retrieval conference (ISMIR)*, pp. 423–428.

[129] Magnusson, T., 2009. *Epistemic Tools: The Phenomenology of Digital Musical Instruments*. Ph.D. thesis, University of Sussex.

[130] Magnusson, T., 2009. Of epistemic tools: Musical instruments as cognitive extensions. *Organised Sound*, 14(02):168–176.

[131] Magnusson, T., 2011. ixi lang: a SuperCollider parasite for live coding. In *Proceedings of International Computer Music Conference*, pp. 503–506. University of Huddersfield.

[132] Magnusson, T., 2011. Algorithms as scores: Coding live music. *Leonardo Music Journal*, 21:19–23.

[133] Mäkelä, M., 2007. Knowing through making: The role of the artefact in practice-led research. *Knowledge, Technology & Policy*, 20(3):157–163.

[134] Malt, M. and Jourdan, E., 2008. Zsa. Descriptors: a library for real-time descriptors analysis. *Sound and Music Computing, Berlin, Germany*.

[135] Manoury, P., 1990. *La note et le son*. L'Hamartan.

[136] Manoury, P., 2013. Compositional Procedures in Tensio. *Contemporary Music Review*, 32(1):61–97.

[137] Manoury, P., 2016. List of works. URL `http://brahms.ircam.fr/philippe-manoury#works_by_date`.

[138] Marino, G. and Santarcangelo, V., 2013. The enaction of conduction: Conducted improvisation as situated cognition. In *SysMus13, Sixth International Conference of Students of Systematic Musicology*, pp. 7–12.

[139] Mazzola, G., 2010. *Musical performance: a comprehensive approach: theory, analytical tools, and case studies*. Springer Science & Business Media.

[140] McCartney, J., 1996. SuperCollider: a new real time synthesis language. In *International Computer Music Conference*.

[141] McCormack, J., 1996. Grammar based music composition. *Complex systems*, 96:321–336.

[142] Mills, P. F., van der Steen, M. M., Schultz, B. G., and Keller, P. E., 2015. Individual differences in temporal anticipation and adaptation during sensorimotor synchronization. *Timing & Time Perception*, 3(1-2):13–31.

[143] Miranda, E. R. and Braund, E., 2017. Experiments in Musical Biocomputing: Towards New Kinds of Processors for Audio and Music. In *Advances in Unconventional Computing*, pp. 739–761. Springer.

[144] Moelants, D. and McKinney, M., 2004. Tempo perception and musical content: What makes a piece fast, slow or temporally ambiguous. In *International Conference on Music Perception and Cognition*.

[145] Moles, A. A., 1972. *Théorie de l'information et perception esthétique*. Denoël, Gonthier.

[146] Montecchio, N. and Cont, A., 2011. A unified approach to real time audio-to-score and audio-to-audio alignment using sequential montecarlo inference techniques. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 193–196. IEEE.

[147] Mouton, R. and Pachet, F., 1995. The symbolic vs. numeric controversy in automatic analysis of music. In *Artificial Intelligence and Music (IJCAI—95 Workshop Program Working Notes)*, pp. 32–40.

[148] Müller, M., Mattes, H., and Kurth, F., 2006. An Efficient Multiscale Approach to Audio Synchronization. In *ISMIR*, pp. 192–197. Citeseer.

[149] Müller, M., Konz, V., Scharfstein, A., Ewert, S., and Clausen, M., 2009. Towards Automated Extraction of Tempo Parameters from Expressive Music Recordings. In *International Symposium on Music Information Retrieval (ISMIR)*, pp. 69–74.

[150] Niedermayer, B., 2009. Improving Accuracy of Polyphonic Music-to-Score Alignment. In *Proceedings of the 10th International Society for Music Information Retrieval Conference*, pp. 585–590. Kobe, Japan.

[151] Niedermayer, B., 2012. *Accurate audio-to-score alignment: data acquisition in the context of computational musicology*. Ph.D. thesis, JKU Linz.

[152] Nouno, G., Cont, A., Carpentier, G., and Harvey, J., 2009. Making an orchestra speak. In *Sound and Music Computing*. Porto, Portugal. SMC2009 Best Paper Award.

[153] Núñez, A., 2012. Toward a Listening Based Taxonomy for Live Electronic Processing of Sound. Case study: Works produced at LIEM. In *Electroacoustic Music Studies Network Conference*.

[154] Orio, N. and Déchelle, F., 2001. Score following using spectral analysis and hidden Markov models. In *ICMC: International Computer Music Conference*, pp. 1–1.

[155] Orio, N., Lemouton, S., and Schwarz, D., 2003. Score Following: State of the Art and New Developments. In *Proceedings of the 2003 Conference on New Interfaces for Musical Expression*, NIME '03, pp. 36–41. National University of Singapore, Singapore, Singapore.

[156] Otsuka, T., Nakadai, K., Takahashi, T., Ogata, T., and Okuno, H. G., 2011. Real-time audio-to-score alignment using particle filter for coplayer music robots. *EURASIP Journal on Advances in Signal Processing*, 2011:2.

[157] Papiotis, P., 2010. *Real-time Accompaniment Using Lyrics-Matching Query-by-Humming*. Ph.D. thesis, Universitat Pompeu Fabra.

[158] Pineda, J. A., 2005. The functional significance of mu rhythms: Translating "seeing" and "hearing" into "doing". *Brain Research Reviews*, 50(1):57–68.

[159] Poncelet, C. and Jacquemard, F., 2015. Model Based Testing of an Interactive Music System. In *ACM/SIGAPP Symposium On Applied Computing*. ACM, Salamanca, Spain.

[160] Puckette, M., 1995. Score following using the sung voice. In *Proceedings of the International Computer Music Conference*, pp. 175–8.

[161] Puckette, M., 1997. Pure data. In *International Computer Music Conference*. Thessaloniki, Greece.

[162] Puckette, M., 2002. Using Pd as a score language. In *International Computer Music Conference*. Gothenburg, Sweden.

[163] Puckette, M., 2004. A divide between 'compositional' and 'performative' aspects of Pd. In *First International Pd Convention*. Graz, Austria.

[164] Puckette, M., 2009. *A case study in software for artists: Max/MSP and Pd*, pp. 119–134. Editions Hyx.

[165] Puckette, M. and Lippe, C., 1992. Score following in practice. In *Proceedings of the International Computer Music Conference*, pp. 182–182. INTERNATIONAL COMPUTER MUSIC ACCOCIATION.

[166] Rabiner, L. and Juang, B., 1993. *Fundamentals of Speech Recognition*. Prentice-Hall Signal Processing Series: Advanced monographs. PTR Prentice Hall.

[167] Raffel, C. and Ellis, D. P., 2015. Large-scale content-based matching of MIDI and audio files. In *Proceedings of the 16th International Society for Music Information Retrieval Conference*, pp. 234–240.

[168] Ramirez, R. and Vamvakousis, Z., 2012. Detecting Emotion from EEG Signals Using the Emotive Epoc Device. *Brain Informatics: International Conference*, pp. 175–184.

[169] Ramona, M., Cabral, G., and Pachet, F., 2015. Capturing a Musician's Groove: Generation of Realistic Accompaniments from Single Song Recordings. In *Proceedings of the 24th International Conference on Artificial Intelligence*, pp. 4140–4141. AAAI Press.

[170] Raphael, C., 1999. Automatic segmentation of acoustic musical signals using hidden Markov models. *IEEE transactions on pattern analysis and machine intelligence*, 21(4):360–370.

[171] Raphael, C., 2010. Music Plus One and Machine Learning. In *International Conference on Machine Learning (ICML)*.

[172] Riff, M. C., Bonnaire, X., and Neveu, B., 2009. A Revision of Recent Approaches for Two-dimensional Strip-packing Problems. *Eng. Appl. Artif. Intell.*, 22(4-5):833–837.

[173] Risset, J.-C., 1999. Composing in real-time? *Contemporary Music Review*, 18(3):31–39.

[174] Ritter, M., Hamel, K., and Pritchard, B., 2013. Integrated multimodal score-following environment. In *International Computer Music Conference*. Ann Arbor, MI: Michigan Publishing, University of Michigan Library.

[175] Roads, C., 1977. Composing Grammars (2nd ed. 1978). In *International Computer Music Conference*.

[176] Roads, C., 1996. *The computer music tutorial*. MIT press.

[177] Roads, C., 2004. *Microsound*. MIT Press.

[178] Roads, C., 2015. *Composing Electronic Music: A New Aesthetic*. Oxford University Press.

[179] Roads, C. and Wieneke, P., 1979. Grammars as representations for music. *Computer Music Journal*, pp. 48–55.

[180] Roberts, C. and Kuchera-Morin, J., 2012. *Gibber: Live coding audio in the browser*. Ann Arbor, MI: Michigan Publishing, University of Michigan Library.

[181] Roberts, J. C., 2007. State of the art: Coordinated and multiple views in exploratory visualization. In *International Conference on Coordinated and Multiple Views in Exploratory Visualization*, pp. 61–71. IEEE.

[182] Robertson, A. and Plumbley, M., 2007. B-Keeper: A beat-tracker for live performance. In *International Conference on New interfaces for musical expression (NIME)*, pp. 234–237. ACM.

[183] Rodriguez-Serrano, F., Vera-Candeas, P., and Carabias-Orti, J. J., 2015. A RealTime Score Follower using Spectral Factorization and Online Time Warping

(Tempo forward version). In *Music Information Retrieval Evaluation eXchange (MIREX)*.

[184] Roederer, J. G., 2008. *The physics and psychophysics of music: an introduction*. Springer Science & Business Media.

[185] Roma, G. and Herrera, P., 2013. *Representing Music as Work in Progress*, pp. 119–134. IGI Global, Hershey, PA.

[186] Romani Picas, O., 2014. *A novel audio-to-score alignment method using velocity-driven DTW*. Master's thesis, UPF.

[187] Rowe, R., 1993. *Interactive Music Systems: Machine Listening and Composing*. MIT Press (MA).

[188] Rowe, R., 2004. *Machine musicianship*. MIT press.

[189] Sandillon-Rezer, N.-F., 2013. Clustering for categorial grammar induction. In *Workshop on High-level Methodologies for Grammar Engineering@ ESSLLI 2013*, p. 13.

[190] Schickore, J., 2014. Scientific Discovery. In E. N. Zalta, ed., *The Stanford Encyclopedia of Philosophy*. Spring 2014 edn. URL `http://plato.stanford.edu/archives/spr2014/entries/scientific-discovery/`.

[191] Schutt, R. and O'Neil, C., 2013. *Doing data science: Straight talk from the frontline*. O'Reilly Media, Inc.

[192] Sebeok, T. A., 1975. Six species of signs: Some propositions and strictures. *Semiotica*, 13(3):233–260.

[193] Sebeok, T. A., 2001. *Signs: An introduction to semiotics (2nd ed.)*. University of Toronto Press.

[194] Shannon, C. E., 1948. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55.

[195] Shneiderman, B., 1996. The eyes have it: a task by data type taxonomy for information visualizations. In *IEEE Symposium on Visual Languages*, pp. 336 – 343. IEEE.

[196] Smith, J. O., 2007. *Mathematics of the Discrete Fourier Transform (DFT)*. W3K Publishing. URL `https://ccrma.stanford.edu/~jos/mdft/`.

[197] Smith, J. O., 2011. *Spectral audio signal processing*. W3K Publishing. URL `https://ccrma.stanford.edu/~jos/sasp/`.

[198] Spiegel, L., 1992. An alternative to a standard taxonomy for electronics and computer instruments. *Computer Music Journal*, 16(3):5–6.

[199] Stark, A. M., 2011. *Musicians and machines: Bridging the semantic gap in live performance*. Ph.D. thesis, Queen Mary.

[200] Stark, A. M. and Plumbley, M. D., 2009. Real-time chord recognition for live performance. In *International Computer Music Conference (ICMC)*.

[201] Stark, A. M. and Plumbley, M. D., 2012. Performance following: Real-time prediction of musical sequences without a score. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(1):190–199.

[202] Stark, A. M., Davies, M. E., and Plumbley, M. D., 2009. Real-time beat-synchronous analysis of musical audio. In *International Conference on Digital Audio Effects (DAFx)*, pp. 299–304.

[203] Stockhausen, K., 1971. Four criteria of electronic music.

[204] Stroppa, M., 1999. Live electronics or...live music? Towards a critique of interaction. *Contemporary Music Review*, 18(3):41–77.

[205] Sturm, B. L., Roads, C., McLeran, A., and Shynk, J. J., 2009. Analysis, visualization, and transformation of audio signals using dictionary-based methods. *Journal of New Music Research*, 38(4):325–341.

[206] Thöle, R., 2008. *Approximation Algorithms for Packing and Scheduling Problems*. Ph.D. thesis, Christian-Albrechts-Universität zu Kiel.

[207] Thompson, E. and Varela, F. J., 2001. Radical embodiment: neural dynamics and consciousness. *Trends in cognitive sciences*, 5(10):418–425.

[208] Toiviainen, P., 2001. Real-time recognition of improvisations with adaptive oscillators and a recursive Bayesian classifier. *Journal of New Music Research*, 30(2):137–147.

[209] Tormene, P., Giorgino, T., Quaglini, S., and Stefanelli, M., 2009. Matching incomplete time series with dynamic time warping: an algorithm and an application to post-stroke rehabilitation. *Artificial intelligence in medicine*, 45(1):11–34.

[210] Toro-Bermúdez, M., Desainte-catherine, M., and Baltazar, P., 2010. A Model for Interactive Scores with Temporal Constraints and Conditional Branching. In *Journées de Informatique Musicale*.

[211] Trapani, C. and Echeveste, J., 2014. Real Time Tempo Canons with Antescofo. In *International Computer Music Conference*, p. 207. Athens, Greece.

[212] Tufte, E., 1990. *Envisioning Information*. Graphics Press.

[213] Väljamäe, A., Steffert, T., Holland, S., Marimon, X., Benitez, R., Mealla, S., Oliveira, A., and Jordà, S., 2013. A review of real-time EEG sonification research. In *International Conference on Auditory Display (ICAD)*. Georgia Institute of Technology.

[214] Vercoe, B., 1984. The synthetic performer in the context of live performance. In *International Computer Music Conference (ICMC)*.

[215] Vos, J. and Rasch, R., 1981. The perceptual onset of musical tones. *Perception & psychophysics*, 29(4):323–335.

[216] Wang, G., 2008. *The chuck audio programming language. a strongly-timed and on-the-fly environ/mentality*. Princeton University.

[217] Wang, S., Ewert, S., and Dixon, S., 2015. Compensating for asynchronies between musical voices in score-performance alignment. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 589–593.

[218] Wiggins, G., Miranda, E., Smaill, A., and Harris, M., 1993. Surveying musical representation systems: A framework for evaluation. *Computer Music Journal*, 17(3):31–42.

[219] Winkler, T., 2001. *Composing interactive music: techniques and ideas using Max*. MIT press.

[220] Wishart, T., 1986. Sound symbols and landscapes. In *The language of electroacoustic music*, pp. 41–60. Springer.

[221] Worrall, D., 2009. An introduction to data sonification. *The Oxford Handbook of Computer Music*, pp. 312–33.

[222] Xenakis, I., 1992. *Formalized Music (2nd ed.)*. Pendragon Press.

[223] Xiong, B. and Izmirli, O., 2012. Audio-to-audio alignment using particle filters to handle small and large scale performance discrepancies. In *International Computer Music Conference (ICMC)*.