REFLEX:
The Viable System Model in the Design of an Interactive Music System

Simon Weins
MSc Sound Design
The University of Edinburgh
2021

Contents

List of Figures

1 Introduction

This thesis explores the application of the Viable System Model (VSM) to the design of an Interactive Music System in Max/MSP. The VSM is an analytical approach to the organization of exceedingly complex structures developed by Cybernetician Stafford Beer. What I will argue throughout this thesis is that the VSM provides a valuable framework for developing a multi-level, modular structure in the development of Interactive Music Systems.

After giving a short introduction to some terminology and historical context that seems necessary for an understanding of the VSM, I will describe the general purpose, the recursive structure, as well as the different sub-systems of the model and their relationship to external environments. At the end of the section, I will briefly discuss the aesthetic relevance of the VSM in a musical context, as well as previous work related to this topic.

After that I will focus on the application of two recursive layers of the VSM to the Interactive Music System REFLEX, which was developed in the visual programming language Max/MSP. I will explain different strategies for low-level adaptation in Digital Signal Processing (DSP), the use of audio descriptors and machine listening and the notion of *Composing Interactions* for the development of cybernetic musical structures. Finally, I will discuss a performance setup which was used to record a number of works with the system.

2 The Viable System Model

2.1 Overview

The VSM is a holistic approach to the organization of complex systems designed by Cybernetician Stafford Beer [1]. The model is based on the structure of the human nervous system and was first described in his publishing of *Brain of the Firm* [2]. The VSM grew out of Beer's analysis that most organizations[1] did not possess enough flexibility to adapt to the fast-moving pace of their environments. Beer concluded that in order for a system to be viable, it must be self-regulatory and able to quickly adapt to changing external pressures.

2.2 Structure

Beer's modelling of the VSM was directly inspired by the work of psychiatrist and fellow Cybernetician Ross Ashby, in particular his work on Self-Regulation, that was later included in what came to be known as *Ashby's Law of Requisite Variety* (see Appendix A.1). Ashby's Law states that in order for a system to be able to adapt to its environment, the control mechanisms of the system must be greater than or equal to the states the system can attain [3]. This meant that the number of possible states of a system must also be equal or greater than the number of possible states of its environment. In order for any system to reach this amount of variety, feedback and recursive control structures between its different parts are needed.

[1] Beer generally considered any closed system, regardless of size or function, as an organisation [2]

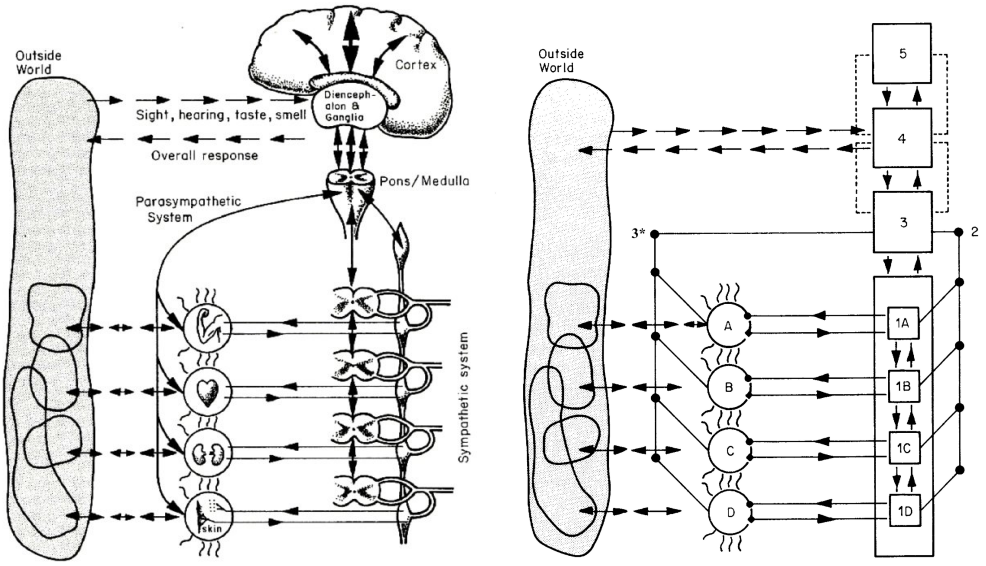

Figure 1: The human nervous system and its adaptation to the VSM in Brain of the Firm [2].

Beer took Ashby's insight and looked for the most complex structure known to humans at the time. The system would have to adapt to a wide range of environments, while still keeping a coherent identity. The various parts of this system had to have large degrees of autonomy while securing the viability of the system as a whole. The system would need to work in a recursive manner to respect Ashby's Law. The human nervous system was the structure Beer used to lay the theoretical grounds of his model [2]. This can be seen in detail in Figure 1.

To make the VSM work in practice, Beer categorized the different aspects of the system and ensured their recursive connectivity. This structural recursion is the primary difference to traditional models of regulation [2]. Beer was convinced that in order for a system to deal with exceedingly complex conditions, each level of recursion had to share the same basic structure and regulatory functions of a general VSM, even if it was functionally different. This fact is illustrated in Figure 2. The recursive structure allowed the VSM increased variety, while securing its overall coherence.

To fully grasp the structure of the VSM, it is crucial to understand the interaction between ENVIRONMENT, OPERATION and MANAGEMENT. The ENVIRONMENT in the VSM can be anything outside the system that the system interacts with. The OPERATION , dealing with Systems 1-3 in Figure 1, takes information and resources from the ENVIRONMENT and transforms them. It deals with the *"Here & Now"* of the system. MANAGEMENT, composed of Systems 3-5 concentrates on the *"There & Then"* by monitoring future environments and ensuring the cohesion of the systems identity in relation to any changes that might occur within the ENVIRONMENT.

2.2.1 Primary Activities

The PRIMARY ACTIVITIES function, also known as System 1, is responsible for the transformation of incoming information. It takes inputs directly from the ENVIRONMENT and transforms them. These can be thought of as divisions in a company or as DSP in the realm of digital information. What is particular about the PRIMARY ACTIVITIES is that they are largely autonomous and self-sustaining. They are in fact a complete VSM in themselves, with an autonomous OPERATION and MANAGEMENT. Usually there are multiple PRIMARY ACTIVITIES happening at the same time, which might share the same resources and ENVIRONMENT. It is therefore crucial that the PRIMARY ACTIVITIES are coordinated so that conflicts or unnecessary labour can be avoided.

2.2.2 Coordination

The COORDINATION function, also known as System 2, is responsible for scheduling and managing the information flow between the PRIMARY ACTIVITIES and higher MANAGEMENT. It is therefore important that COORDINATION uses a common language that is understood by the different departments. This can be done via normalization of values. Beer proposes the normalization of values between 0-1 [2]. Correct COORDINATION is in place if oscillation in the OPERATION can be prevented. This function is always working to ensure the autonomy of the PRIMARY ACTIVITIES while sustaining the larger whole.

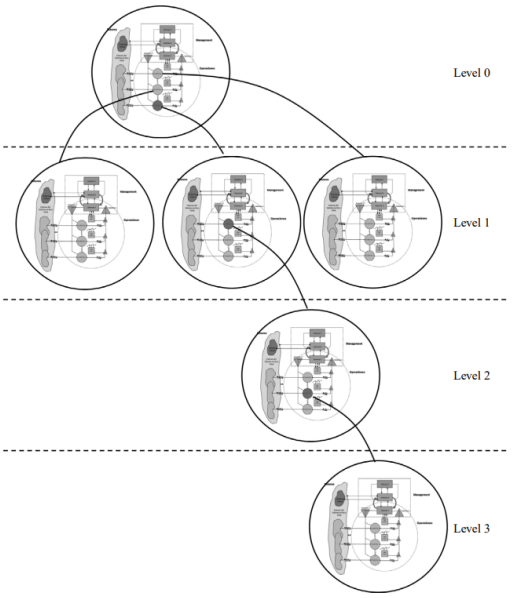

Figure 2: Recursive structure of the VSM from Perez [4].

2.2.3 Cohesion & Audition

The COHESION function, split up in System 3 and System 3*, in many ways is at the heart of the VSM as it ensures the adaption of the whole system by balancing interactions in the OPERATION and acting as an interface to higher MANAGEMENT. It is largely responsible for monitoring the PRIMARY ACTIVITIES, and stepping in if their self-regulation can not be maintained. COHESION receives information directly from the local MANAGEMENT of the PRIMARY ACTIVITIES and from COORDINATION. It also receives information from the AUDITION function, also known as System 3*, which infrequently communicates with the PRIMARY ACTIVITIES directly without receiving information from local MANAGEMENT. This is done in an effort to show care for the PRIMARY ACTIVITIES, but also to make sure that the filtering system of the local MANAGEMENT is accurate.

It is crucial to note, that COHESION should only deal with residual variety of the PRIMARY ACTIVITIES, but should never be responsible for their regulation. If this happens, the OPERATION is likely to end up in a continuous oscillation between PRIMARY ACTIVITIES, COORDINATION and COHESION . It is therefore important that COHESION only enforces minimal direct commands and should mostly act indirectly by distributing appropriate measures.

2.2.4 Intelligence

The INTELLIGENCE function, also known as System 4, directly observes the ENVIRONMENT and reacts to trends and possible threats to the viability of the system. It is largely responsible for developing adaptation plans to unforeseen circumstances that can be executed by the OPERATION. Generally it ensures that the system can react rapidly to changes in the ENVIRONMENT by being in constant interaction with the COHESION of the internal OPERATION.

2.2.5 Policies

The POLICIES function, also known as System 5, is responsible for ensuring that the identity and values of the system are kept. It ensures that the whole organization stays cohesive by monitoring the interaction between COHESION and INTELLIGENCE . It comes only rarely into force, but has the power to steer the whole system into a completely different direction.

2.3 Discussion

The effectiveness of the VSM has been proven in a number of different contexts, ranging from cooperations to governmental structures [5]. However, what is the usefulness of the VSM in an aesthetic and particularly a musical context?

The application of the VSM to the musical field has previously been described by Darren Pickles. In his pieces *Sin-Plexus* (2013) and *The Beast* (2015) Pickles uses audio-rate feedback from a microphone, long delay lines and other DSP to control the amplification and dampening of an incoming signal via Ableton Live [6]. While Pickles implements solutions for the OPERATION, his implementation of higher-level MANAGEMENT is not automated or clearly defined. Furthermore Pickles only describes one recursive layer of his system and thereby misses the importance of the recursive structure inherent to the VSM [4]. In that sense his performance system is not hugely different from other open, audio-rate controlled feedback systems [7].

Nonetheless, Pickles provides a detailed description of how the VSM can be applied to musical systems and offers many useful analogies for the different subsystems of the VSM in a musical context. A valuable contribution is also given by Pickles' *Cybernetic Compositional Framework* [6]. Pickles' notes that a cybernetic musical work is *ephemeral, performative, non-notated, decentralized, eco-systemic, interactive, recursive* and *revealing of true 'being'*. Without going into too much detail, particularly about the last point, it is easy to see that any musical work produced with the VSM will probably follow this description.

A structural and functional resemblance to the VSM can also be found in other prominent examples of Interactive Music Systems that use control data feedback or Machine Listening techniques (see Appendix A.2). Interesting connections could also be made to networked music groups such as *The Hub* in which control data is exchanged between different agents [8]. In this context it is particularly interesting to note that technology might become the ENVIRONMENT itself [9]. However, a detailed analysis of this is beyond the scope of this thesis. Last, but not least it is also worth pointing out that Brian Eno was greatly influenced Beer's Brain of the Firm in his development of what later came to be known as Ambient music [10].

3 REFLEX - an Interactive Music System

3.1 Overview

REFLEX is an interactive music system developed in the visual programming language Max/MSP. Following Rowe's classification of Interactive Music Systems, the system could be described as a *performance-driven, transformative Player* [11]. Using a number of DSP solutions, an external audio source is used as trigger and input to create textured sounds, syncopated rhythms and expanded counterpoint. The system was developed, following a modular structure by separating DSP and Graphic User Interface (GUI) and developing a coherent module size based on available screen space similar to the Jamoma project [12]. An overview of the GUI of REFLEX in its current implementation can be seen in Figure 3.

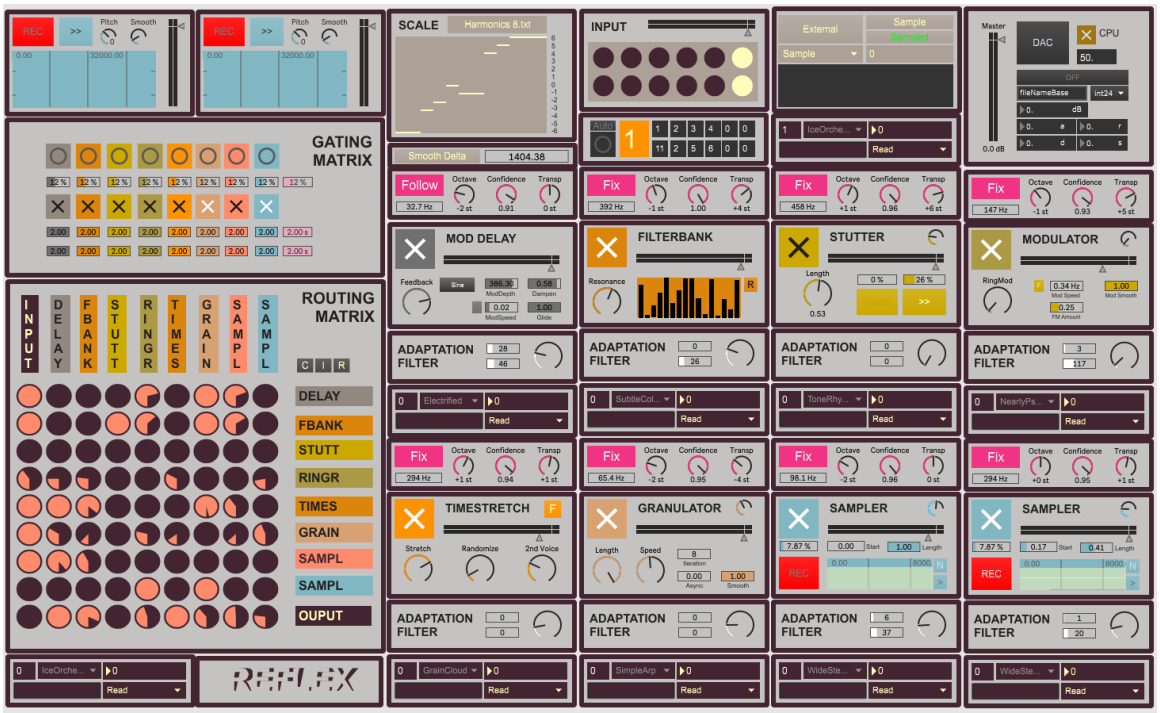

Figure 3: Graphic User Interface of REFLEX.

3.2 Structure

The structure of REFLEX was developed by focusing on two recursive layers of the VSM. The first layer describes the top-level performance system and the second layer deals with what could be described as the *Adaptive Signal Processing* layer of the system. The different parts of each layer and their relationship to the ENVIRONMENTcan be seen in full detail in Figure 4. A guiding principle in the design of the system was to maximise its variety so that it could be employed in a number of different contexts. This was mainly achieved by considering the low-level adaptation of each DSP module, but also by the implementation of a matrix mixer that could influence trigger decisions in

the top-level OPERATION. MANAGEMENT in the top-level structure was facilitated by using a preset system, audio descriptors and machine learning techniques to influence parameter settings.

3.2.1 Adaptive Signal Processing

Different DSP solutions were implemented in the Adaptive Signal Processing layer of the system to deal with its PRIMARY ACTIVITIES. The different DSP solutions were chosen based on their sonic identity and in order to extend the variety of the system. The final implementation contains the following modules:

- **Mod Delay**. Time-variant delay-line that can be modulated or used as a Plucked String Model [13].
- **Filterbank**. Transposable filterbank with 16 resonant bandpass filter.
- **Stutter**. Circular buffer of 1,000 ms that copies a selected group of last received samples. Produces coloured clicks, glitchy artefacts and electronic tones similar to Waveset or Wavetable Synthesis.
- **Modulator**. Complex FM ring modulator.
- **Timestretch**. Timestretch algorithm using delay lines.
- **Granulator**. Zero-crossing granulator [14].
- **Sampler**. Speed-variant playback of a recorded section of up to 10,000 ms.

After each of these DSP modules an FFT Filter is implemented. The Filter subtracts the amplitude of FFT bins from the unprocessed signal and thereby prevents the accumulation of frequencies in areas of the spectrum throughout the system. In practice, the module leads to complex and interesting behaviours ranging from simple sidechaining to modulations and glitchy high-frequency artefacts that shape much of the systems sonic identity. The amount of adaptation and smoothing between the different bins is adjustable as can be seen in the *SpectralSubtraction Demo* patch. While other modes for self-regulation in the DSP modules were considered (see Appendix A.3 and A.4), this method proved to be the most flexible and reliable.

To deal with the accumulation of high-frequencies from the filter an additional low-pass filter was inserted after each FFT. The low-pass filter is based on the Moog Ladder design and sets in after each module after a certain frequency threshold is crossed. An estimate for the threshold is determined by analysing the Spectral Centroid, a measure for evaluating the overall brightness of the spectrum [15]. COORDINATION between the Spectral Centroid and the filter is regulated via homeostatic mapping (see Appendix A.3) which automatically adjusts the cutoff frequency of the filter once the threshold is crossed. A detailed demonstration of homeostatic mapping can be seen in the *Homeostat Demo* patch.

COHESION in the modules was achieved through the regulation of their DSP parameters. With the exception of the low-pass filter, all parameters can be set manually and saved as a preset. Some parameters of each module are modulated by a set of descriptors that are used as the INTELLIGENCE

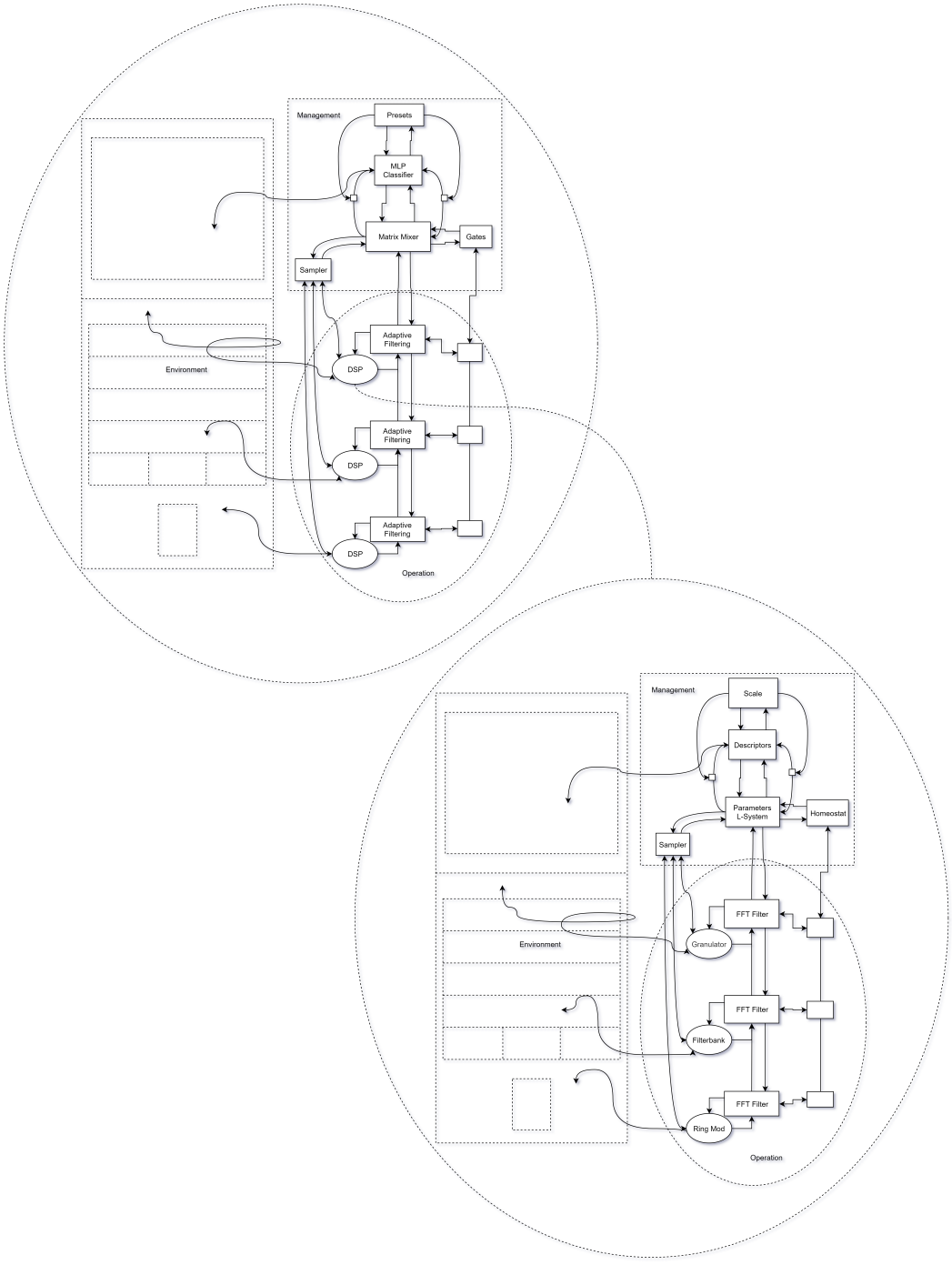

Figure 4: Two recursive layers of the VSM in the REFLEX system.

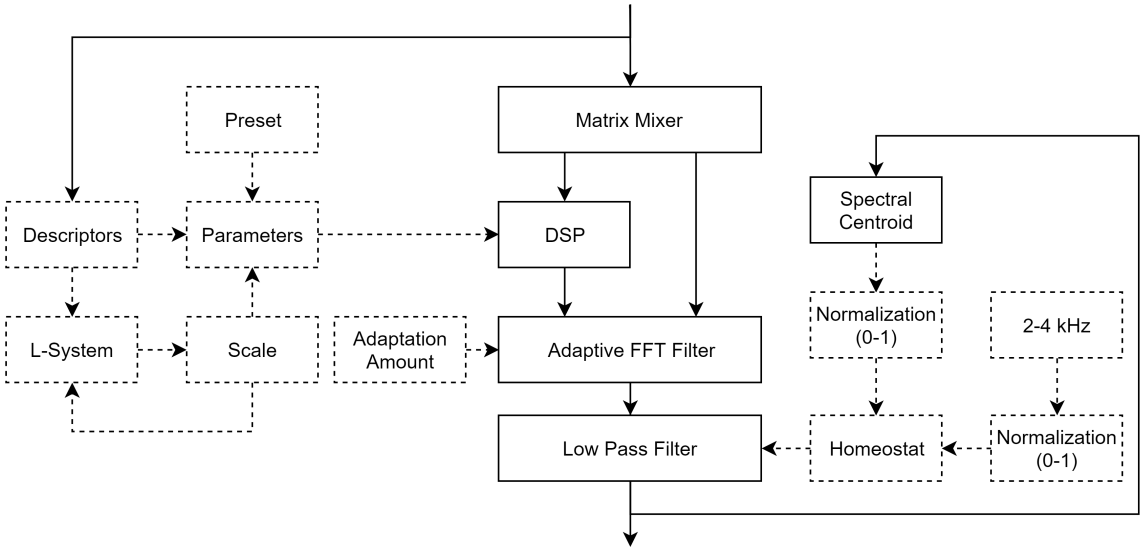

Figure 5: Schematics of an adaptive DSP module.

function. Currently only onset-detection, pitch-estimation and delta-time are measured from the ENVIRONMENT and sparsely used for parameter changes in the DSP. However, many more custom-made descriptors were developed (see Appendix A.4). The current set of descriptors were developed with the FluCoMa Toolkit [16]. In a future iteration a routing matrix could be used to assign various descriptors for parameter modulation. First experiments in this direction can be seen in the *Descriptor Modulation Demo* patch.

The transposition and pitch range of each module can be manually chosen from a range of microtonal and equal tempered scales which determine the POLICIES function for each module. The scales currently used were developed using the *Scala* software [17] and the *Scale Workshop* online tool [18]. Once a set of notes in a scale is selected, the information is fed into an L-System (see Appendix A.5) that coordinates the transposition of notes in each module. Figure 5 shows a schematic of all the modules involved in the adaptive behaviour of each DSP solution.

3.2.2 Circular Triggering

COORDINATION in the top-level system is achieved via a gating matrix that receives triggers from the PRIMARY ACTIVITIES to decide whether a sound should be turned on or off. Each trigger symmetrically turns a voice on or off using the ADSR~ envelope generator. The probability for each voice to be turned on or off can be set manually via the %gate function by Gregory Taylor [19]. Figure 6 shows a schematic of this function in its current implementation. Alternative models for controlling the density were tried out, such as controlling the connectivity amount from the matrix mixer described in section 3.2.3 as well as controlling the density amount via an L-System Density Function [20] (see Appendix A.5). However, due to the symmetric response of the trigger, a simple heuristic measure proved to be the simplest solution. The circular interconnectivity of these gates opening and closing upon receiving a trigger creates complex rhythmic structures and emergent behaviour

that shapes much of the fragmented aesthetic of the system.

3.2.3 Routing & Resampling

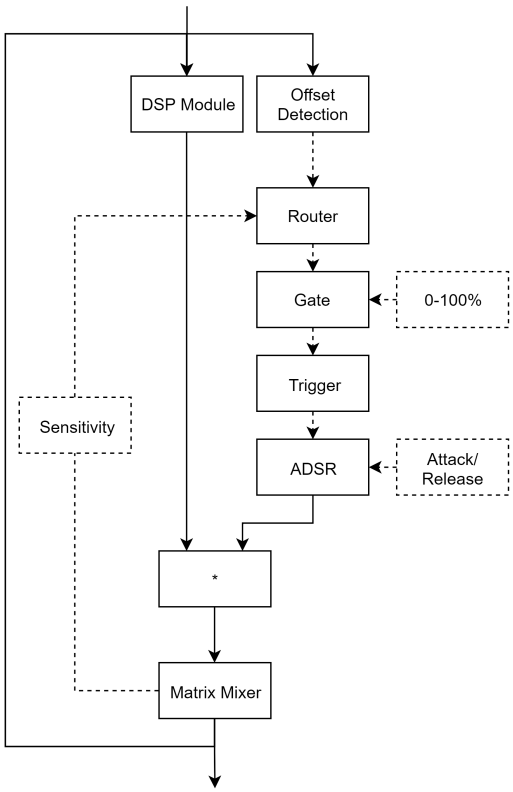

Figure 6: Circular Triggering.

To create COHESION between the PRIMARY ACTIVITIES and the ENVIRONMENT a 9x9 matrix mixer is implemented to route an external sound source through the network of DSP modules. The matrix permits incremental amounts of DSP outputs to be routed into other DSP inputs as well as into their own input. This allows for a great amount of different timbres to be generated by routing the DSP in different ways. The connection between modules also determines the trigger connectivity of the COORDINATION function described in section 3.2.2. This linkage between COORDINATION and COHESION acts on the sound in a nearly sculptural way, producing different sonic perspectives of each sound source.

Two global samplers at the output of the matrix act as the AUDITION function in which up to 60 seconds of audio can be recorded. The signal of these can be fed back into the main input of the matrix and played back at different speeds. Currently this implementation is incredibly simple. However, the function extends the variety of the system significantly by allowing for extended repetitions of recorded material.

3.2.4 Machine Listening

For the INTELLIGENCE function a multi-layer perceptron (MLP) classifier was implemented to detect different external sound-sources by writing peaks of Mel-Frequency Cepstral Coefficients (MFCC) into a dataset. The framework for the classifier was developed by Owen Green as part of the FluCoMa project [21]. Further logic surrounding the descriptor and the dataset were developed and make it possible to add new sounds to the dataset and to write, recall and train the classifier on the fly as can be seen in the *Classifier Demo* patch. Once trained, the classifier can predict different sound sources with relatively high accuracy. In its current implementation, the classifier is used to recommend preset states described in the following subsection. This makes it possible to change instruments during a performance. The classifier can then automatically match a suitable preset for each sound source.

It must be noted that other examples for implementations of the INTELLIGENCE function, such as the *Checo* program in Beer's *Cybersyn* project often include some kind of prediction model for *long-term future* developments [2]. This is currently not the case as data is still processed in real time. Therefore further exploration in this direction is still needed to develop a more sophisticated approach for this function.

3.2.5 Composing Interactions

For the POLICIES function a global preset system was implemented to have access to most of the visible parameters of the system. The presets can be linearly interpolated, giving the potential to smoothly transition from one state to another over a given period of time. This solution proved to be very effective in acting as the POLICIES function, without unnecessarily restricting the systems variety. The careful design of these presets can be used to compose intricate interactions in the system [22].

3.3 Discussion

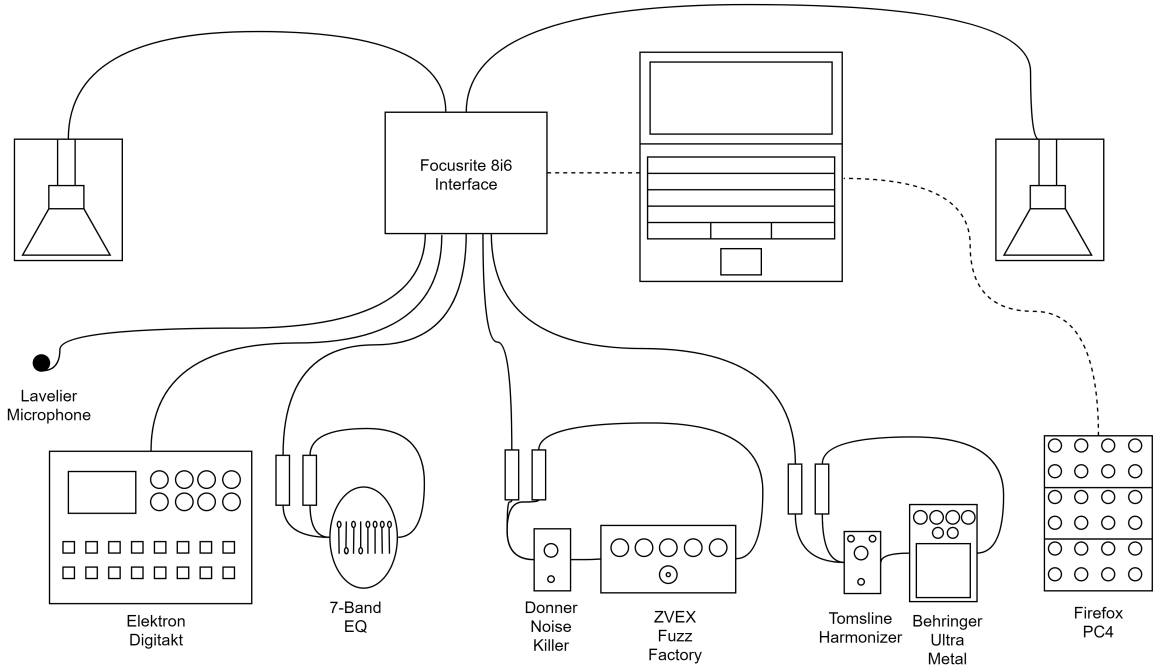

Figure 7: Performance Setup.

Several experiments with different sound sources were conducted to be used as ENVIRONMENT for the system. These included simple feedback circuits made by feeding analogue effect pedals back into themselves, using Lavelier microphones for the amplification of small objects and working with pre-sampled sounds. In the final implementation, all these sound-sources were used as can be seen in Figure 7. The classifier in the INTELLIGENCE function was then trained on the different sound

sources and could react to changes of a soundsource. This seemed particularly useful for longer performances with the system as can be seen in the ClassDemo.mp4 video example.

However, each routing already provides an incredibly rich world of complex interactions that can be explored for extended periods. A number of recordings were made that explore the "misbehaviour" of Machine Listening [23] and the aesthetics of digital artefacts such as clicks and aliasing [24] in each routing. The system was mainly played by controlling the resampling function, the gate probability and the preset selection of each DSP modules as well as the external sound source.

A selection of these recordings can be found in the *Audio Recordings* folder in which I explored some of these artefacts, such as the production of clicks through various DSP solutions (Scrubbing 1.wav), odd melodies being played through inaccurate pitch recognition (Scrubbing 2.wav), audio feedback of the matrix mixer (Scrubbing 3.wav), resampling of short looping cycles (Scrubbing 5.wav), rhythmic modulation through extreme usage of the adaptation filter (Scrubbing 7.wav) as well as several of these artefacts interacting at the same time.

In future iterations of REFLEX, I would be curious to extend the recursive structure of the system to the next lower and higher recursion level. This would include a structure of the VSM on a Microsound level[1], as well as the inclusion of the current top-level system in a networked structure, possibly processing input signals from another person or system. While some efficiency improvements can be made, the top-level structure of the REFLEX is robust, flexible and offers enough variety to be used in various musical contexts.

4 Conclusion

As demonstrated, the VSM provides a valuable structure for the development of an Interactive Music System. The relative independence of the VSM subsystems allow for a modular approach to designing smaller units that can easily be extended or changed in future iterations. The metaphors of ENVIRONMENT, OPERATION and MANAGEMENT in the VSM suit themselves particularly well for the development of a system in Max/MSP as analogue-to-digital-conversion, MAX Signal Processing and MAX control data map perfectly on the different areas of the model. Finally, the recursive structure of the VSM offers a useful perspective on structuring musical material on different perceptual and temporal levels, using the same structure, but accommodating the peculiarities of each level at the same time.

[1]The development of the Zero-Crossing Granulator and some exploration of GenExpr was a first step in that direction [25]

References

[1] Espejo, Raul. *The Viable System Model.* Systemic Practice and Action Research 3 (1990).

[2] Beer, Stafford. *Brain of the Firm.* 2nd ed. Classic Beer Series, 1995.

[3] Ashby, W. Ross. *An Introduction to Cybernetics.* 2nd ed. Methuen, 1976.

[4] Pérez Ríos, José. *Models of Organizational Cybernetics for Diagnosis and Design.* Kybernetes 39, no. 9/10 (2010).

[5] Pickering, Andrew. *The Cybernetic Brain: Sketches of Another Future.* University of Chicago Press, 2011.

[6] Pickles, Daren. *Cybernetics in Music.* Coventry University, 2016.

[7] Sanfilippo, Dario, and Andrea Valle. *Feedback Systems: An Analytical Framework.* Computer Music Journal 37, no. 2 (2013).

[8] Gresham-Lancaster, Scot. *The Aesthetics and History of the Hub: The Effects of Changing Technology on Network Computer Music.* Leonardo Music Journal 8 (1998).

[9] Haworth, Christopher. *Ecosystem or Technical System? Technologically-Mediated Performance and the Music of The Hub.* Proceedings of the Electroacoustic Music Studies Network Conference. Berlin, 2014.

[10] Eno, Brian. *Generating and Organizing Variety in the Arts.* Studio International 192, no. 984 (1976).

[11] Rowe, Robert. *Interactive Music Systems: Machine Listening and Composing.* MIT Press, 1994.

[12] Place, Tim, and Trond Lossius. *Jamoma: A Modular Standard for structuring Patches in Max.* International Computer Music Conference. New Orleans, 2006.

[13] Karjalainen, Matti, Vesa Valimaki, and Tero Tolonen. *Plucked-String Models: From the Karplus-Strong Algorithm to Digital Waveguides and Beyond.* Computer Music Journal 22, no. 3 (1998).

[14] Sanfilippo, Dario. *Granulation without Windowing Using Delay Lines and Zero-Crossing Detection.* Dariosanfilippo.tumblr.com. Accessed August 12, 2021. https://dariosanfilippo.tumblr.com/post/184802756176/granulation-without-windowing-using-delay-lines.

[15] Peeters, Geoffroy. *A Large Set of Audio Features for Sound Description (Similarity and Classification) in the CUIDADO Project.* IRCAM. Paris, 2004.

[16] Tremblay, Pierre Alexandre, Owen Green, Gerard Roma, and Alex Harker. *From Collections to Corpora: Exploring Sounds through Fluid Decomposition,* International Computer Music Conference, 2019.

[17] *Scala.* Computer software. www.huygens-Fokker.org. Accessed August 12, 2021. http://www.huygens-fokker.org/scala/index.html.

[18] *Scale Workshop.* Computer software. sevish.com. Accessed August 12, 2021. https://sevish.com/scaleworkshop/.

[19] Taylor, Gregory. *Step by Step: Adventures in Sequencing with Max/MSP.* Cycling '74, 2018.

[20] Bell, Renick. *Experimenting with a Generalized Rhythmic Density Function for Live Coding.* Linux Audio Conference, 2014.

[21] *FluCoMa: Quick Audio Classifier.* Vimeo, 2021. https://vimeo.com/561582497.

[22] Di Scipio, Agostino. *'Sound Is the Interface': From Interactive to Ecosystemic Signal Processing.* Organised Sound 8 (2003).

[23] Bowers, John, Green Owen. *Hijacking Listening Machines for Performative Research.* New Interfaces for Musical Expression, Virginia Tech Blacksburg, 2018.

[24] Cascone, Kim. *The Aesthetics of Failure: 'Post-Digital' Tendencies in Contemporary Computer Music.* Computer Music Journal 24, no. 4 (2000).

[25] Roads, Curtis. *Microsound.* Cambridge, MIT Press, 2004.

[26] Eldridge, Alice. *Collaborating with the Behaving Machine: Simple Adaptive Dynamical Systems for Generative and Interactive Music.* University of Sussex, 2008.

[27] Lewis, George E. *Too Many Notes: Computers, Complexity and Culture in Voyager.* Leonardo Music Journal 10 (2000).

[28] Ciufo, Thomas. *Beginner's Mind: An Environment for Sonic Improvisation.* In Proceedings of the ICMC, 2005.

[29] Hsu, William. *Strategies for Managing Timbre and Interaction in Automatic Improvisation Systems.* Leonardo Music Journal 20 (2010).

[30] Ashby, W. Ross. *Design for a Brain.* Springer Netherlands, 1960.

[31] Pask, Gordon. *A Comment, a Case History and a Plan.* In J.Reichardt (ed.), Cybernetics, Art, and Ideas. New York Graphic Society, 1971.

[32] Grill, Thomas. *Constructing High-Level Perceptual Audio Descriptors for Textural Sounds.* Proceedings of the 9th Sound and Music Computing Conference; Kopenhagen, 2012.

[33] Winkler, Todd. *Composing Interactive Music: Techniques and Ideas Using Max.* MIT Press, 2001.

[34] Manousakis, Stelios. *Musical L-Systems*, The Royal Conservatory The Hague, 2006.

[35] *Fastest Onset Detection? (Native or External).* cycling74.com/forums. Accessed August 12, 2021. https://cycling74.com/forums/fastest-onset-detection-native-or-external.

[36] *Pattrstorage with Umenu for Naming Presets?.* cycling74.com/forums. Accessed August 12, 2021. https://cycling74.com/forums/pattrstorage-with-umenu-for-naming-presets.

[37] Parker, Martin. *healthyMaxing.* github.com/tinpark. Accessed August 12, 2021. https://github.com/tinpark/healthyMaxing/tree/master/$_{t}opLevelPatchingSuite$.

A Appendix

A.1 Ashby's Law of Requisite Variety

Ross Ashby was a British psychiatrist, who played a leading role in the development of Cybernetics in the UK [5]. His work in this field was focused on the adaptation of complex systems. He saw that one of the major problems with systems being unable to adapt, was that they were unable to deal with the variety present in their environment. Within that context, Ashby defined the term variety as *all possible states that a system can take* [4]. He concluded that the variety of a system must be equal or greater than the variety of its environment. This seemingly simple and obvious observation proofed to have serious implications: In order for a system to deal with the changes in an environment, the system must have *all* the information available in the environment, including *all* information about itself, as being a part of this environment and even *all* information about this information. Seemingly an impossible task.

In practicality it meant, being able to react in a manner that is appropriate to a given situation without having to know all the details of the environment. In this context, being aware of the internal workings of the system also gives information about the external environment. The problem Ashby faced was how to achieve great variety with minimal resources. His solution was to breaking down an input into smaller bits, let different operations deal with their transformation and distribute information about their states inside the system. This came later to be known as the homeostat.

To illustrate the problem, imagine an environment that has 50000 possible states. If a given system was a single entity, this entity would need to know and monitor these 50000 states at all times in order to respond in an appropriate manner. It goes without saying that this would be an enormous task for a single operation. If, however, this operation was split into 3 smaller operations and these operations were in a recursive exchange about their activities, each operation would only have to cover 10 states each to reach requisite variety.

$$3^{10} = 59049$$

Let us assume that the previously mentioned environment was part of a larger system with greater variety. A following step to react to this increased variety would be to embed our previous three operations into a larger system that is complemented by another system that also has three operations, each dealing with 10 different aspects of the environments and all systems being connected in a recursive fashion. Our system would now be able to cover an astronomic amount of variety, closer to the variety of a real complex environment:

$$3^{100} = 5.1537752e + 47$$

This highlights the importance of feedback in a system that is aiming for requisite variety on the one hand, but also shows that it is possible to reach large amounts of variety with limited resources. In fact, each operation can deal with incredibly simple tasks, giving it greater autonomy, but also making it much easier to identify possible problems.

A.2 Analysing Interactive Music Systems with the VSM

The VSM offers a powerful tool for analysing and comparing Interactive Music Systems. Rowe defines interactive music systems as *"those whose behavior changes in response to musical input"* [11].

As such, all interactive music systems explore the interaction between what is happening inside and outside of a given system and this is exactly what the VSM is trying to do as well. Naturally the most prominent works in the field already exhibit a large structural resemblance to the VSM on at least one recursion level:

- **Alice Eldridge - *Self-karaoke Machine***. Eldridge's example uses a granular synth as PRIMARY ACTIVITIES function. The synth is fed samples from an external ENVIRONMENT. COHESION is provided by a homeostat and a simple physics simulation. The latter can also be said to act as INTELLIGENCE function as it influences long term structural behaviour of the granular synth [26]. The POLICIES function is not clearly assignable in this case.
- **George Lewis - *Voyager***. Lewis' implements different MIDI-based algorithms as a possible COHESION function to order musical material played by up to 64 synthesised voices used to deal with the PRIMARY ACTIVITIES. INTELLIGENCE is provided by analyzing a 4-7 second long analysis window from an external ENVIRONMENT. POLICIES are regulated by three performance modes in which an algorithm decides to react to the external sound source by either ignoring, contrasting or adapting to what is being played outside the system [27].
- **Thomas Ciufo - *Beginners Mind***. Ciufo's implementation offers a multi-layered recursive analysis of input and output material of his system. The used descriptors for the analysis regulate the INTELLIGENCE function. COHESION is reached via a matrix mixer that regulate the distribution of DSP modules as PRIMARY ACTIVITIES which transform an incoming sound source. For his implementation of POLICIES, Ciufo implements a preset system similar to the implementation in REFLEX [28].
- **William Hsu - *ARHS***. Hsu uses an analysis window of 1000ms for his INTELLIGENCE function. Various descriptors send information to the POLICIES function which regulates the COHESION of speed, brightness and noisiness of an FM synth used as PRIMARY ACTIVITIES that responds to timbral changes in the ENVIRONMENT [29].

The aspect that is lacking in most of the above is the recursive structure that is inherent to the VSM. None of the examples *really* develop any low-level adaptation strategies and there are good reasons for this. Generally processing power can become a real problem when there are many DSP solutions and multiple audio-channels involved. Furthermore, sometimes the strength of a system lies in its simplicity. Nonetheless, this is one of the interesting challenges and potentials when working with the VSM. Aside from this it is hard to clearly assign an AUDITION and COORDINATION function in all of the above. More detailed insight in most of the systems would be needed to assess whether the functions are present or not.

A.3 Homeostatic Mapping

Homeostatic Mapping as seen in Figure 8 formed the core of much of the low-level adaptation that I first implemented in REFLEX. The basic idea is that DSP parameters are changed based on low-level analysis of the DSPs output, forming a feedback loop between the change in the sound and its control parameters. This idea is inspired by Ross Ashby's implementation of his homeostat [30], which has successfully been implemented before to the musical field in various projects by Alice Eldridge [26]. The homeostat that I implemented for my system takes an analysis value from a descriptor such as the Spectral Centroid in the range of 0-1 and compares it to an ideal state, a

criterion of stability (COS).

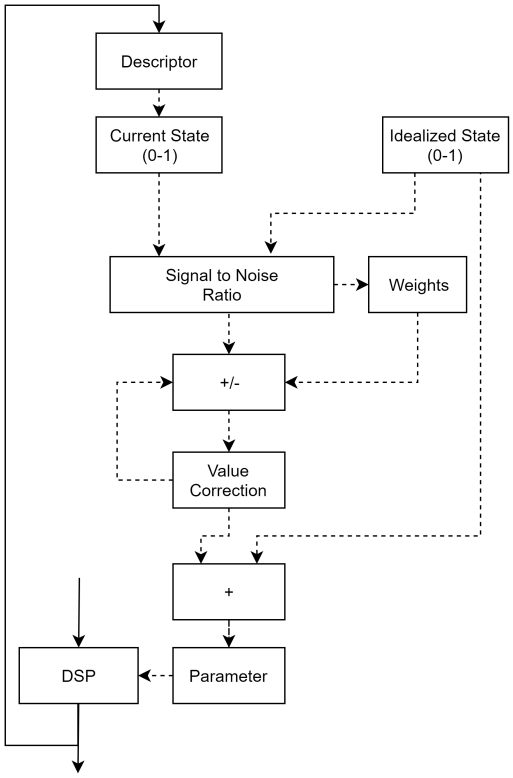

Figure 8: Homeostatic Mapping.

The homeostat is incredibly efficient in finding a good balance point when the parameter maps accurately to the descriptor as can be seen in the *Homeostat Demo* patch in which a Spectral Centroid is used with a sine wave oscillator. In this case the Spectral Centroid is exactly the same as the frequency of the oscillator.

In the first prototype of the system, all DSP parameters were controlled by monitoring analysis values of the DSPs output. While this produced interesting results, the problem was that the parameter tended towards the extremes within a given parameter range, resulting in constant uncontrollable chaos. This didn't seem very interesting or practical, a fact that was also noted by Cybernetician Gordon Pask [31]:

> It is clear that an aesthetically potent environment [...] must offer sufficient variety to provide the potentially controllable variety [in Ashby's terms] required by a man (however, it must not swamp him with variety - if it did, the environment would be merely unintelligible).

The problem was often that the descriptors did not match the parameters they were controlling so accurately that they could be used with the homeostat. An example of this can be seen in the *Alt DSP Adaptation Demo* patch in which a variable Zero-Crossing windowing is used to granulate an incoming signal. Various descriptors to describe density, speed and "orderliness" of the signal were matched with parameters that regulate the grain-length, playback speed and asynchronicity of the grains. While - on surface - this seems like a sensible mapping, the relationship between the DSP parameter and the descriptors tends towards the extremes of the spectrum (either 0 or 1). This is due to the fact that the descriptors do not perfectly describe the parameter space for which they are used, resulting in the problem of oscillation between COORDINATION and COHESION described in section 2.2.3.

Overall there is still potential with this mode of self-regulation, but it involves extreme fine-tuning of the descriptors and a lot of patience in getting the parameters just right to produce reliable results.

A.4 Audio Descriptors and Machine Listening

Different high-level perceptual audio descriptors were developed throughout the project, although only few of them were used in the final implementation. For a full view of all the descriptors that were developed, please open the *Descriptor Demo* patch.

While there are many libraries for low-level descriptors in Max/MSP, it was difficult to find good descriptors for structural developments and phrase detection. Grill proposes a defined parameter space for the construction of high-level perceptual audio descriptors for textural sounds that seemed suitable to my purposes [32]. Loosly based on Grill's taxonomy, descriptors for the following parameter spaces were developed:

1. **Quiet-Loud** using Root Mean Square (RMS) average
2. **High-Low** brightness estimation by using the Spectral Centroid
3. **Tonal-Noisy** spectral estimation by using a measure for Tonalness and Spectral Flatness
4. **Smooth-Coarse** envelope estimation by comparing current RMS to RMS of previous analysis window
5. **Fast-Slow** speed estimation by detecting the delta-time between events using onset-detection
6. **Ordered-chaotic** rhythmicity estimation by comparing the regularity of delta-times
7. **Sparse-Full** estimation to detect the "percussiveness" or "sustainedness" of a sound in a given analysis window

The spectral parameter-spaces 1-3 as well as the use of deltatime between events for an overall speed estimation follow relatively common methodologies [15] [33], the estimates for *Ordered-chaotic*, *Smooth-Coarse* and *Sparse-Full* were more experimental in nature.

These parameter spaces were simply calculated by using an onset/offset-detection of the signal. The detection is relatively accurate up to a speed of about 14-15Hz. Beyond this threshold, the detection of separate events becomes blurry. Generally the consideration of an appropriate threshold and analysis window are crucial for good results. A threshold of 0.5 for the onset-detection produced good results in most situations. An analysis window of 1500ms seemed to be a good compromise between speed and accuracy for most descriptors.

The parameter space Ordered-chaotic was calculated by comparing the last 8 delta-times between onset-detections. The decision to take 8 values is somewhat arbitrary, but seemed to strike a good balance between precision and speed. The list of delta-times is then processed by detecting how many of the values repeat in the list. Repeating values are excluded from the list. The shorter the list, the higher the synchronicity. This value is multiplied by the difference between the highest and lowest value in the list and dividing it by the average length of the delta-time. If this value is high, asynchronicity tends to be high as well. For more accurate results, a certain roughness for the delta-time could be considered to make up for such things as off-notes or swing. Further precision could be implemented by considering local symmetries in the list, e.g. to consider if a phrase is played at double or half speed - something that is very common in rhythm-based music.

The parameter space Smooth-Coarse is used as an envelope evaluation by detecting the logarithmic attack/decay time of a sound. The value was calculated by subtracting the averaged RMS of the analysis window from the current RMS and taking its absolute value. If the value is high, the envelope tends to be sharp. If the value is low, the attack tends to be smoother. This is very crude, but works relatively well when the threshold value is well adapted.

Sparse-Full was calculated by detecting the Temporal Centroid of the signal by considering the amount of time that the signal is below or above a given threshold. The value is then averaged over the analysis window. This tends to give some indication if the current material being played contains more sustained or percussive material.

In a future iteration of REFLEX, I would like to develop these descriptors so that they could be used for the modulation of parameters. This would allow for extended possibilities of using various descriptors without the restriction of them having to perfectly match to the parameter space. A first demo of this can be seen in the *Descriptor Modulation Demo* patch in which various descriptors are used to influence the parameter space of a simple FM Synth from the gen~ examples folder.

A.5 Musical L-Systems

For the implementation of note transposition in each DSP module a Lindenmayer-System (L-System) was implemented based on a design that was developed for the second semester of Non-Real-Time Systems. L-Systems are designed to model growth in biological processes using recursive application of a limited set of rules [34]. For the transposition of notes in the system, simple rules for the modification of an initial list are fed into the algorithm. The list is then updated by replacing current values with values from the rule-set, creating self-similar behaviour and producing simple melodies with a restricted set of notes being played.

In REFLEX, the system provides a new value each time a pitch-estimation from the fluid.pitch~ external of the Flucoma toolkit is estimated to be above a set pitch-confidence level. This has the effect that a new transposition is chosen each time that threshold is crossed. As the rule-set for each DSP module and its pitch-confidence estimation differs, complex counterpoints, chords and harmonies can be created from a limited set of selected notes.

A similar approach to using L-Systems as simple logic that produces complex results can also be seen in Renick Bell's *Generalized Rhythmic Density Function* [20]. Bell's approach was used to develop a Table L-System Density Function for the second semester of Non-Real-Time Systems that would create complex poly-rhythmic structures. The function was initially used in REFLEX to distribute the trigger probability. However, the symmetric response of the Circular Triggering described in section 3.2.2 made this function imperceptible and was therefore replaced by a much cheaper function using randomization. In a future iteration of REFLEX, I would be curious to expand its possibilities for use with more metric and rhythmic material. In this context the described L-System Density Function might be much more useful.

A.6 Dependencies and Abstractions in REFLEX

For the implementation of REFLEX, I used a number of objects from the FluCoMa Toolkit [16] as well as the following third party abstractions:

- **%gate** is based on a patch with the same name by Gregory Taylor [19].
- **onsetdetection** is based on an abstraction by *Rodrigo Constanzo* [35].
- **Pattrstorage_Manager** is based on an abstraction by user *Floating Point* from the Cycling74 forum [36].
- **Master_Recorder** is based on the mp.quickRecord abstraction by Martin Parker [37].
- **SpectralCentroid~** is based on the gen~.centroid patch from the gen~ examples folder.
- **gen~ Pitchshifter** is based on the gen~.pitchshift patch from the gen~ examples folder.
- **FM Synth** in the *Descriptor Modulation Demo* patch is from the gen~.fm_bells abstraction in the gen~ examples folder.