

CONTENTS

Introduction.....	2
Structure	3
Prediction.....	3
Schemas	3
Behaviours.....	4
Time delays	5
Implementation.....	5
Input.....	7
Analysis.....	7
Prediction.....	8
Manager	9
Concatenative synthesis	9
VST effect	10
Output.....	11
Usage.....	11
Initialisation.....	12
Configuration.....	12
Editing schemas.....	12
Editing behaviours	13
Time delays	14
Future Developments	14
Evaluation.....	14
Optimisation	15
Extension.....	15
Appendix – Hierarchy of subpatches and abstractions	15
Bibliography	15

INTRODUCTION

Currently, performance-oriented music software tends to be aligned with one of two directions: on the one hand, artificial agents modelled on a human performer, and on the other, human-controlled sound production based on sampling, synthesis, processing or combinations thereof.

My design, which I call *Tim*, is an attempt to combine the two approaches, both in the matter of decision-making and in that of sound production. The aim is to facilitate free improvisation by providing the human performer with an adaptive, dynamic soundscape which complements their playing and stimulates creativity. In this sense, Tim is designed to neither be a fully-fledged improv partner nor a simple audio effect, but something in-between.

A major influence in my research was David Plans-Casal and Davide Morelli's improv agent, *Frank* (Plans-Casal & Morelli, 2007). Similarly to Frank, Tim draws control information as well as audio material from the human improvisator's live signal.

Furthermore, I found the work of William Hsu on his *London* and *AHRS* improv systems (Hsu, 2008) to be of great value. Ideas about how AHRS deals with anticipating the musical intent of the performer and adapting over large periods of time have been an inspiration for Tim's multilayered design.

Essentially, Tim functions as follows: a pool of material is collected into a database, and descriptive information is synthesised sequentially from the input signal. This information is then dynamically handled by a system of *schemas* and *behaviours*.

As in (Huron, 2006), I use the term "schema" to denote an expectational "set" based on previous experience. To this end, Tim uses an artificial neural network (ANN) to model dependencies within audio sequences, allowing for *prediction*. The predicted results and the inherent errors then affect how the system behaves; either by responding sympathetically with the human's playing, or by trying to induce contrast or surprise. Thus, the relation between schema and behaviour is not simply of cause and effect: each can prompt a change in the other dynamically.

The user can choose to have a varying degree of control over the creation of schemas and behaviours. Two extreme possibilities appear: one in which the musician configures a set of schemas and corresponding behaviours prior to the performance as well as potentially loading pre-existing material into the database, and one in which the settings are random and the pool of sonic material is gradually filled up during the performance. Of course, combinations in-between are also possible, and finding a satisfying balance for a specific situation can prove to be an appealing challenge for the musician.

The final stage in Tim's signal flow deals with the performer's temporal expectations. Based on a psychological experiment which studied the way subjects adapt to progressive delaying of visual feedback to physical stimuli (Cunningham, Billock, & Tsou, 2001), a similar strategy is employed here in the audio domain. Depending on the chosen parameters, the effect can prove more or less challenging for the improvisator to adapt to, thus pulling them out of their comfort zone and

impelling them to constantly re-examine their temporal relation to the sonic feedback produced by Tim.

STRUCTURE

As previously established, the system's operation is based on three main elements: input sequence prediction, expectational schemas, behaviours and time delays. It is time to take a closer look at each of them and learn what they entail.

PREDICTION

The basic mechanism of step-by-step prediction enables the overall dynamic quality of the system. Periodically, a feature vector consisting of four numeric descriptors is fed into an ANN, which attempts to predict the values of the next feature vector, and so on. The feature vector describes the input signal in terms of timbre (brightness, noisiness, loudness) and time (onset density). To make sure that the system can adapt to virtually any sound source, all four descriptors are scaled relative to the running minimum and maximum values.

At the start of the activity of one of Tim's schemas, the ANN is cleared and taken into the training phase. Each previous incoming feature vector F_{t-1} is paired up with the current target vector F_t . This enables the system to adapt to the current input signal from the musician. Once a certain number of patterns have been fed into the network and the training phase is over, it begins anticipating future vectors F_{t+1} according to the current F_t .

As previously suggested by (Collins, 2008), a pragmatic approach to achieving an effect of anticipation and adaptation is to use predictive success as the deciding factor. As long as the neural network is reasonably successful (the maximum allowed error being a parameter of the schema) at predicting incoming descriptor values, the current schema is considered valid. Once the predictive error exceeds the allowed threshold, a different schema is chosen from the available options.

There is a musical justification for this type of interaction: human improvisors tend to act in a more deterministic fashion while in a comfortable situation. It is expected that in the course of moving on to a different situation, their produced input becomes incompatible with the ongoing predictive schema, thus prompting a change in the system's state.

However, a convincing human-machine interactive model also needs to work in the reciprocal direction. The system is to be more than merely reactive to the human's activity; it should also actively bring about new situations for the human to react to. As the following sections show, the dynamic relationship between schemas and behaviours allows Tim to perform in this way.

SCHEMAS

We have already defined the main function of schemas: using a predictive model to assign a state to the system, relative to the human improviser's current activity. The schemas however do not determine Tim's sonic output directly; instead they each hold a probability distribution over a set of output behaviours.

Thus, schemas are similar to states in a Hidden Markov Model (HMM), with the difference being that these states are not part of a Markov process: rather than a set of transitional probabilities between schemas, transitions from one schema to another are conditioned by the traits of the musician's input signal.

Each schema has up to three activation conditions, each specifying one of the four sonic features extracted from the input (brightness, noisiness, loudness, or onsets) and the corresponding admitted minimum and maximum incoming percentages. As long as the input signal meets the schema's set of conditions, an activation request is periodically broadcasted. When the time comes for a schema change, one of the ongoing active requests is then randomly accepted.

Another purpose of the mediation between input and output by way of schemas is the establishment of long-range dependencies throughout a performance. A major interest of many free improvisors is the opportunity for call-backs and variations on previously produced material. In (Plans-Casal, 2008), the author shows how 'short-circuiting' parts of his algorithm helped improve the temporal connection between computer and musician. Conversely, (Hsu, 2008) extends his design with a statistical "performance mode" analysis module, responsible for long-term decisions.

Inspired by both of these methods, I approached this problem through the handling of schema changes: they are meant to be as spontaneous as necessary to stimulate variety, while at the same time being of a sufficiently deterministic nature as to allow for consistency and familiarity over time.

It has been established how changes in the musician's activity can prompt a schema change, by way of the predictive network. Let us now consider the bilateral relationship between Tim's schemas and behaviours.

BEHAVIOURS

Returning to the HMM analogy, behaviours are the equivalent of the system's observable output. Behaviour activation is determined by the probability distribution within the currently active schema. Each of the behaviours can be assigned a time limit, or last indefinitely. In the former case, the conclusion of the behaviour's activity will prompt the system to switch schema, which in turn will result in the activation of a new behaviour.

Figure 1 shows a block diagram of the structural elements mentioned so far.

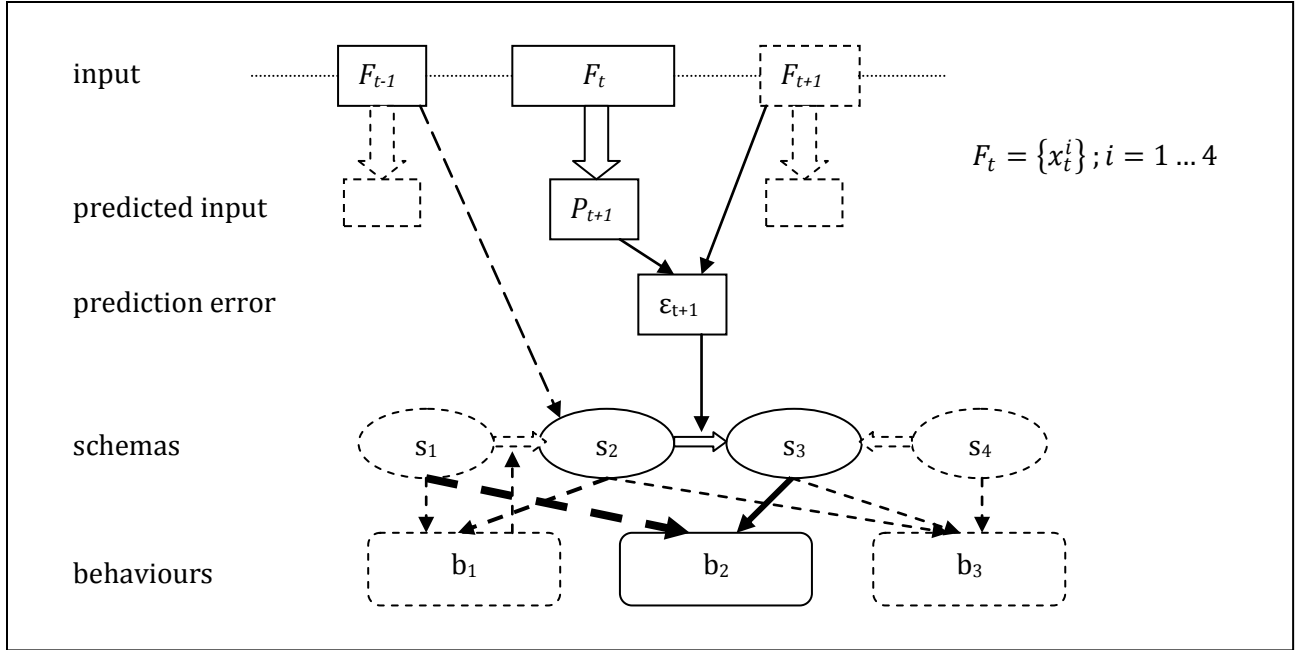


Figure 1. Tim's structural dependencies. Arrow width represents probability. Dashed lines represent potential realisations.

Unlike designs such as Frank and AHRS, Tim does not intend to play the role of a human performer – not even a performer of laptop music. I have aimed to give the system a different character, both in terms of control and the actual sound production.

There are two options for real-time sound generation, each with definable settings particular to the respective behaviour. Two slots are available, functioning in parallel, allowing for any combination. The two existing ways in which Tim treats sound are concatenative synthesis and VST effect processing. Each will be explored more in-depth in the next chapter.

TIME DELAYS

Inspired by the visual experiment in (Cunningham, Billock, & Tsou, 2001), Tim can apply a cycle of up to 10 different delay times, each phase lasting for a specified number of seconds before moving on to the next one.

An interesting implication is what the experiment's authors call *negative after-effect*: once a subject has adapted to a certain delayed feedback setting, removing the discrepancy causes a reduction of the subject's accuracy and agility. In fact, some subjects even reported a reversal between perceived cause and effect. Tim's delay cycles suggest the possibility of transferring this experience to a musical setting.

IMPLEMENTATION

Tim has been developed using Max/MSP, the visual programming language for music and multimedia developed by Cycling '74. Max's modular nature favours encapsulation and extension, which enables the design of a system like Tim to be flexible and open.

A Max program consists of a main *patch*, where *objects* are interconnected using *patch cords*. Similarly to functions and variables in traditional programming, Max/MSP objects can store and process different data types and structures. Any patch can contain a number of embedded *subpatches* or *abstractions*, which in turn can be part of a deeper hierarchy. Data can be transmitted from one subpatch to another by use of dedicated send/receive objects.

Figure 2 is an example of a simple subpatch which computes a mathematical operation between the latest three values coming from its *inlet* 1, and sends the result out its *outlet* 1.

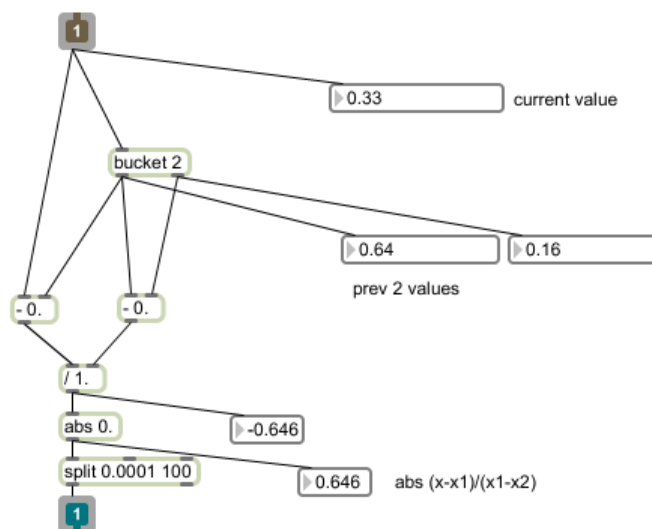


Figure 2. Patching in Max/MSP.

This chapter is broken down according to the major modules which make up the system. Figure 3 displays a general overview of the system's architecture.

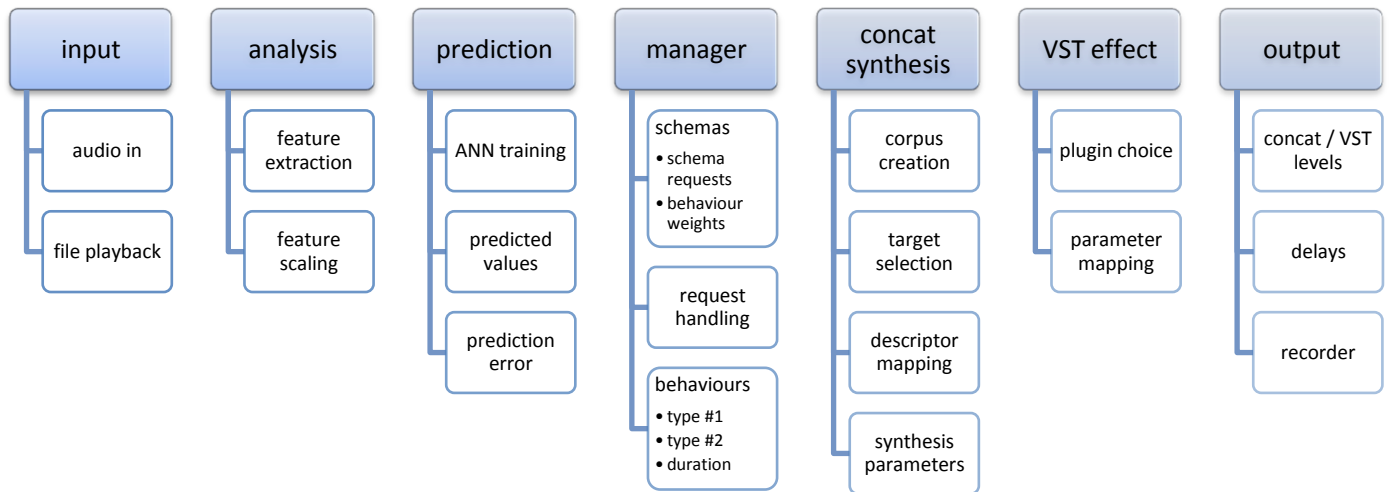


Figure 3. Tim's modular architecture.

INPUT

Tim accepts a monophonic audio signal from one of the soundcard's input channels. Additionally, an audio file may be streamed into the system. The two processes run in parallel and are both controlled by the same gain slider; a more complex routing system was deemed superfluous.

The audio signal is fed into the *analysis* units, as well as both sound processing modules - *concatenative synthesis* and *VST effect*.

ANALYSIS

The extraction of descriptive features from the audio signal is performed by two types of Max structures.

The first is a set of three MSP¹ externals created by Tristan Jehan, freely downloadable from his website (Jehan). `brightness~` and `noisiness~` are FFT²-based objects which measure the spectral centroid and spectral flatness of a signal, respectively. Combined, they offer a general profile of the input signal's timbre at any given time. Meanwhile, `segment~` detects onsets of sound.

The second analysis unit computes the fourth parameter, which again pertains to timbre. *Energy* is calculated using objects from the FTM&Co toolkit which is available online at (Schnell).

The three timbral parameters arrive in a semi-continuous stream of floating point data. In order to make the values manageable and relevant, a running mean is calculated using the Max `mean` object, which is then refreshed with every *tick*.

¹ MSP denotes the part of Max that deals with audio signals, as opposed to discrete numeric or text data. Names of MSP objects end in the `~` character.

² Fast Fourier Transform – algorithm which computes the Discrete Fourier Transform of a signal, representing a running slice of the signal in the frequency domain.

Ticks are *bang* messages being sent out periodically as long as the system is turned on. They trigger events in several areas of the program, including the analysis module. The tick rate varies between 500 ms and 10 seconds; the mechanism for determining its value will be explained in the next section.

The final step is to scale the features to values between 0. and 1., relative to the running minima and maxima. For the timbral parameters this means they can adapt dynamically to the spectral qualities of virtually any input material. As for the onset counter, it becomes a measure of onset density, describing the running amplitude envelope. This was considered preferable to a basic attack counter, which would have also been far more prone to errors caused by false triggering.

PREDICTION

The four scaled features are fed into the prediction subpatch, which is based around a Java implementation of a feed-forward back-propagation artificial neural network for Max/MSP, available online at (Robinson). The advantages and limitations of back-propagation networks have been widely documented in books such as (Rojas, 1996). I considered the speed and accuracy of an ANN with one hidden layer and 4 nodes per layer to be sufficient for the present task.

As expected, there are two phases involving the neural network. The training phase consists of sending two lists containing the previous and the current feature vector into the ANN module, followed by the message `addPat`, thus combining the two vectors into an input-target training pair, or pattern. This process is repeated for a user-defined number of times, after which the object receives the message `train 10000`, which sets the internal weights of the network after 10 thousand train cycles. For a number of 10 input patterns, the training process takes less than a second on a modern computer.

Following is the test phase, in which the trained network predicts the upcoming feature vector based on the incoming list. The predictions are compared with the actual subsequent values, and prediction errors are calculated for each of the four features. An aggregate error is then determined by averaging the three lowest error values. Both this and the individual errors assist in controlling various parameters within the system.

One such parameter is the tick rate, which was mentioned in the previous section. Its initial value is 2 seconds. With each subsequent tick, the current rate is multiplied by a weighted average of the second-order differences between incoming values. Each weight is equal to the inverse of that feature's corresponding error. Thus, the equation becomes:

$$rate_{t+1} = rate_t \cdot \frac{\sum_{i=1}^4 (1 - err_t^i) \left| \frac{x_t^i - x_{t-1}^i}{x_{t-1}^i - x_{t-2}^i} \right|}{\sum_{i=1}^4 (1 - err_t^i)}$$

The effect is of a tick rate that increases for more homogenous chunks of sound, and decreases for sound that varies in timbre and density. This behaviour is meant to simplistically emulate human attention: the more information stimulates our senses, the more alert we become. Using the

prediction errors as weights has the effect of counterbalancing the unpredictable jitter which tends to appear when working with estimative descriptors of audio signals.

MANAGER

The manager is the central module, handling schema change requests and activating the appropriate behaviours. Both schemas and behaviours are loaded within `poly~` objects, allowing them to function in 8 parallel instances each, even though only one of each can be active at any given time. The loading and saving of presets is implemented using `pattrstorage` objects.

All schemas receive the current feature vector at their input. If the features fall within the activation domain, the schema sends out a list containing its ID (a number from 1 to 8), followed by a list of 8 behaviour weights (numbers from 0 to 99). These weights are defined by the user for each schema using a `multislider` object. The requests are delayed by a random interval of up to 20 ms, which ensures that multiple requests arrive in a random order, with no one schema taking precedence over another.

In order to be activated, the request must pass through a `ggate` object. This gate is closed after any successful request, and is opened in case of one of three events:

- at the initialisation of the system;
- in case the prediction error becomes larger than the user-defined maximum;
- when a behaviour has reached the end of its user-defined time limit.

Once a new schema has been chosen, the predictive ANN is cleared, entering a new training phase. Also, the set of new behaviour weights triggers a behavioural switch³. To this purpose, I made use of the probability distribution capabilities of Max's `itable` object.

As well as the optional time limit, behaviours enable the user to control the functioning of the two available sound generation modules. These will be examined in-depth in the following sections.

CONCATENATIVE SYNTHESIS

The concatenative synthesis engine is an essential component of Tim's architecture. It is based around Diemo Schwarz's *CataRT* system (Schwartz) which in turn makes ample use of FTM&Co. The description on the CataRT website explains:

The concatenative real-time sound synthesis system CataRT plays grains from a large corpus of segmented and descriptor-analysed sounds according to proximity to a target position in the descriptor space. This can be seen as a content-based extension to granular synthesis providing direct access to specific sound characteristics.

CataRT is a very powerful and multifaceted system, allowing for interactive exploration of corpora and audio mosaicing. Tim's central extension to CataRT is the way target grains are specified. Normally, the user would manually pick the grains using the mouse or another bi-dimensional

³ of course, the possibility remains that the new schema and/or behavior could be the same as the old ones, depending on the results of the allocation procedures

controller. Tim takes over this role, having the ability to specify target points of up to four dimensions, corresponding to the four components of the sonic feature vectors.

The grain corpus can be initialised with pre-existing sonic material. Regardless of this option, once processing is turned on, the system starts capturing audio grains from the input and adding them to the corpus in real time, roughly two every tick. Both import rate and grain length are thus linked to the tick rate; the grain size is roughly a tenth of the tick period.

For each imported grain, CataRT calculates a vector of 24 descriptors relating to timbre, time or identification within the corpus. Any of these descriptors can be linked to one of the four sonic features of the input signal. To ensure that the mapping is correct, the descriptors are scaled from 0. to 1. similarly to the input features.

The target selection is controlled by two settings specified within the active behaviour. Seeing as how each of the behaviours has two slots, there is a potential maximum of two voices acting in parallel. Each voice can act according to two settings:

- anticipate / follow: using either the predicted feature vector, or the incoming feature vector to specify the target
- supplement / complement: the former leaves the features unchanged, while the latter inverts the feature values

The feature-to-descriptor mapping and the synthesis parameters are also defined on the behaviour level, and their usage will be expanded upon in the next chapter.

As the concept behind Tim is to provide constantly evolving soundscape, it makes sense for the exploration of the corpus to vary continuously, both in terms of *what* and of *when* grains are played. Thus, a variable *grain rate* has been introduced, controlling the rate at which sounds are triggered. Along with the use of `slide` objects, it enables a continuous stream of sound to be generated. With an initial value of 200 ms, the grain rate increases according to the “popularity” of played grains.

Grain popularity is measured by way of a running histogram, monitoring every grain “hit”. In the interest of variety, when a certain grain becomes too popular (i.e. its number of hits is greater than four times the average), one of its neighbouring⁴ grains from the corpus is played instead.

VST EFFECT

Tim’s second option for sound transformation is by way of VST effects. Steinberg’s VST is a widely used standard for audio processing plugins, and the loading and control of VST effects is supported natively by Max/MSP.

As indicated earlier, the two sound generation slots within Tim function in parallel. Thus, rather than acting as inserts within an effect chain, the VSTs act as independent send effects. Consequently, while more traditional effects such as compression and EQ are perfectly applicable,

⁴ in terms of ID.

more interesting results can potentially be obtained when blending a VST effect with a second sound source; be it dry signal, CataRT synthesis, or another VST effect.

Unlike the concatenative synthesis module, the VST framework is situated mostly within the behaviours themselves, communicating with the main patch by way of `send~` and `receive~` objects. This enables the facile switching between effects and settings when going from one behaviour to another, the retaining of behaviour-centric mappings and presets, as well as cross-fading between effects, avoiding unpleasant abrupt changes.

Similarly to the CataRT module, any of the four audio input features can be mapped to one of the parameters of the VST effect. The next chapter shall provide an illustration of this feature.

OUTPUT

In keeping with the principles of transparency and parallelism, the levels for the CataRT and VST modules can be adjusted independently. Note that, at times when there is no VST plugin loaded, the VST slider controls the level of the unprocessed input signal.

The resulting stereo mix is fed into the time delay module. This module contains its own `patrrstorage` object, managing the loading and saving of delay settings independently of schema and behaviour settings. This enables the delayed feedback effect to occur imperturbably, regardless of any developments in the program's state.

The final stereo signal is sent to both the soundcard's output, and into the recorder subpatch. Here, a simple start/stop button allows the user to record Tim's output to a .wav file.

USAGE

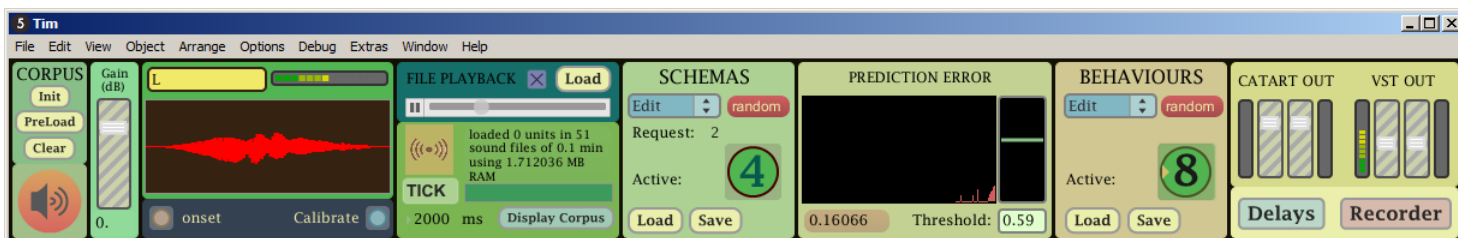


Figure 4. Screenshot of Tim's main panel



Tim exists in two formats: standalone application and Max/MSP collective. The standalone package does not require a Max installation to run. The Max collective requires Max or Max Runtime⁵, and enables the user to open and dissect the patch.

The latest version of Max (5.1.4 at the time of writing) is highly recommended to ensure maximum stability. Both the standalone and the collective versions have been tested on Microsoft Windows-based machines.

⁵ the free, read-only version of Max, available at (Cycling '74)

INITIALISATION

Once the application has been run, the recommended initial sequence of actions is the following:

1. initialise the concatenative synthesis engine by pressing the “Init” button
2. if desired, pre-load material from audio files by pressing the “PreLoad” button – this chops the files up into grains according to pitch analysis, and imports them into the corpus
3. choose an input channel for live audio – by default, this is set to the first (left) channel of the audio interface
4. start the audio engine by clicking the  icon
5. start the CataRT engine by pressing the  button.
6. begin playing live audio into the soundcard and/or streaming a file from the hard drive.

The File Playback box enables wave file loading and playback control. The blue toggle switches between regular and looped playback.

Pressing the “Display Corpus” button opens a timbrally-determined representation of the corpus, with each grain being assigned a point within the two-dimensional space of brightness/periodicity, and colour representing the third dimension, that of loudness.

CONFIGURATION

While the initial settings are sufficient for fully-featured operation, the user has the ability to configure and optimise many aspects of Tim’s performance.

The “Calibrate” button opens a new panel for setting the appropriate levels for onset detection. Depending on the nature of the signal, the default values may be sufficient, or the user could save their own settings as a preset by shift-clicking in one of the available slots. Clicking on a slot recalls a saved setting. Thus, it is possible to have different onset detection settings available, matching the attack patterns of various sound sources.

Schemas and behaviours can be edited by selecting their number from the respective drop-down menus. Additionally, pressing the “random” button randomises the parameters of the whole set of schemas or behaviours. This is useful for quickly coming up with new combinations, though their immediate usability will vary. Next we will take a look at the editing panels of schemas and behaviours, and the parameters they make available.

EDITING SCHEMAS

Next to numbers, schemas can be assigned specific names. While helpful for identifying a certain schema, names have no actual effect on the system. Clicking on a name opens an editing dialog box.

Beneath the schema’s name is a display of the current four scaled features. They can serve as guides to setting the appropriate activation range. The more activation parameters are chosen, the more complex the conditions under which a schema will send out activation requests. Each of the three parameters can be chosen from a drop-down list containing the four features, and their respective ranges are set by clicking and dragging within the accompanying sliders. Green LEDs signal when

individual conditions are met, and a global LED at the bottom lights up when activation requests are sent as a result of all conditions being met.

Two prediction-related parameters are next: train duration determines the length of time between the activation of a schema and the commencing of informed prediction. Naturally, a larger number of training patterns enables a more accurate configuration of the neural network at the cost of the system spending time “in the dark”. The second parameter, error tolerance, specifies the maximum admitted predictive error before the system switches to a different schema.

Finally, weights corresponding to each of the 8 behaviours can be edited using the sliders on the right of the panel. Any combination is possible; e.g. for a 1:1 dependency between schema and behaviour, set all sliders but one to zero.


EDITING BEHAVIOURS

Similarly to schemas, behaviours can be assigned names which can be modified by clicking on them.


The main part of the behaviour panel is taken up by the two sound generation slots. The slots can be set to either CataRT or VST, thus enabling access to their respective sets of parameters.

As discussed in the previous chapter, the concatenative synthesis selection behaves along two coordinates: anticipate/follow and supplement/complement. These are chosen using the first two drop-down lists.

The “Mapping” button opens a new panel allowing the user to pick which specific features are anticipated/followed by Tim, and how they are mapped to the grain descriptors computed by CataRT. The default setting is the conventional two-dimensional combination of brightness – spectral centroid, and energy – energy. More creative options can be explored though, such as e.g. mapping the musician’s noisiness to Tim’s loudness, and countless other configurations of up to four feature-descriptor pairs. The slider labelled “Inertia” controls how closely linked the pairs are, i.e. for large values of inertia, the output descriptors take more time to catch up with the input features.

The second CataRT panel is accessed by the  toggle, allowing the user to choose how the selected grains are handled and transformed by the synthesis object. By modifying sliders and number boxes, the user has control over time and transformation-related parameters and their respective amounts of random deviation. The “Default” button neutralises all the values, thus indicating the playback of grains in their unmodified form.

Choosing the VST option gives access to a similar set of parameters. Firstly, the VST plugin is chosen from a drop-down list of the plugins present in Tim’s folder and subfolders. The default setting is *again.dll*, a simple plugin consisting of a single gain slider, leaving the signal unaffected initially.

The  toggle opens the plugin’s own edit window. Any changes made within the edit window can then be stored and recalled using the “Save” and “Load” buttons. As with the CataRT option, the “Mapping” button opens a panel enabling the user to connect the four input features to any of the

VST plugin’s modifiable parameters, thus overriding the fixed position those parameters had in the edit window. Again, this dependency is subject to “inertia”, as determined by the slider’s position.

Finally, a specific duration can be imposed on the behaviour. Alternatively, if the number box is set to zero, the behaviour will run indefinitely – that is, until a switch is triggered from somewhere else.

TIME DELAYS

The final stage in the signal flow is the cycle of time delays. The panel controlling this module is accessible by pressing the “Delays” button.

Two `multislider` objects allow the user to set the delay times, and the durations of each delay phase in the cycle. Two settings can be stored by clicking on the “Set A” and “Set B” buttons; adjusting the slider underneath makes it possible to interpolate between the two sets of stored values.

A dry/wet slider adjusts the balance between the original and the delayed signals. Seeing as how the delays run continuously from the moment the program starts, the only way to turn it on and off is by adjusting the dry/wet ratio.

The “Load” and “Save” buttons manage the storing and recalling of pairs of A/B settings. Located above them is a display showing the current phase in the cycle, and next to it is a reset button which returns to the start of the cycle, *after* the ongoing phase has run its course.

FUTURE DEVELOPMENTS

While Tim is currently in a fully functional state, as described in the preceding chapters, its modular and transparent design leaves the door open for further work. The potential developments outlined in this chapter are divided into three types: evaluation, optimisation and extension.

EVALUATION

For the most part, the system has been tested privately by me. An early version of Tim was part of my performance at the Making Waves 2 festival (Burloiu, 2010). Also, a number of consultation and testing sessions with Ewan Stefani and Christophe de Bezenac were of major importance in providing me with feedback and suggestions.

At present, Tim is ready for the beta stage, and I plan to make the application available for download, encouraging feedback from interested parties. In the future, a more systematic evaluation covering rehearsal and performance situations could prove highly beneficial. In this sense, the HCI⁶-inspired methods put forward in (Hsu & Sosnick, Evaluating Interactive Music Systems: An HCI Approach, 2009) would be of great help. The guiding principle is that of identifying and quantifying key performance criteria along which performer and audience can then rate their experiences.

⁶ Human-computer interaction

OPTIMISATION

This section pertains to improving Tim's stability and performance. Testing has revealed the integration of CataRT to be at the same time a great strength and a weakness. The power and complexity of CataRT makes it ideal for experimentation and design, but unforeseen combinations can lead to glitches and instability. I have already excised certain superfluous parts of CataRT (such as the mouse-driven selection), and I would like to continue paring down the unnecessary functions, or perhaps even rebuild the concatenative engine from the ground up.

A second area which could be subject to testing and optimisation is the prediction module. The simple feed-forward ANN used within the schemas is sufficient for modelling simple sequences over limited periods of time. However, for large scale time prediction which could determine high-level decisions, a type of recurrent neural network (RNN) would probably be more efficient. A fast and powerful option is the Long Short-Term Memory (LSTM) model, introduced in (Hochreiter & Schmidhuber, 1997). This type of network has been applied to music generation in (Eck & Schmidhuber, 2002) and (Brandmaier, 2008), with promising results.

EXTENSION

While the standalone version of Tim may suffice for many musicians, the ability to use it in conjunction with other effects and processes would open up a range of creative possibilities. Using the Max for Live toolkit, Tim could be adapted to work within Ableton Live. This integration would make Tim subject to Live's sequencing, preset management and effect chaining capabilities.

In terms of functionality, one area in which Tim could be development is behaviour flexibility. For instance, an element of surprise could be introduced to affect behaviour parameters on-the-fly or end a behaviour's activity. This feature could be correlated with the long-term prediction discussed in the previous section, to produce context-relevant changes in behaviour.

APPENDIX – HIERARCHY OF SUBPATCHES AND ABSTRACTIONS

...

BIBLIOGRAPHY

Brandmaier, A. (2008). *ALICE - A LSTM-Inspired Composition Experiment*. Retrieved 2010, from brandmaier.de: <http://www.brandmaier.de/alice/>

Burloiu, G. (Performer). (2010, June 9). *Making Waves 2*. School of Music, University of Leeds, Leeds, UK.

Collins, N. (2008). Reinforcement Learning for Live Musical Agents. *Proceedings of ICMC2008*. Belfast: International Computer Music Conference.

Cunningham, D. W., Billock, V. A., & Tsou, B. H. (2001, November). Sensorimotor Adaptation to Violations of Temporal Contiguity. *Psychological Science*, pp. 532-535.

- Cycling '74. (n.d.). *Download*. Retrieved 2010, from Cycling 74: <http://cycling74.com/downloads/>
- Eck, D., & Schmidhuber, J. (2002). *A First Look at Music Composition using LSTM Recurrent Neural Networks*. Manno: IDSIA.
- Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation* 9(8) , 1735-1780.
- Hsu, W. (2008). Two Approaches for Interaction Management in Timbre-aware Improvisation Systems. *International Computer Music Conference*. Belfast.
- Hsu, W., & Sosnick, M. (2009). Evaluating Interactive Music Systems: An HCI Approach. *NIME*. Pittsburgh.
- Huron, D. (2006). *Sweet Anticipation: Music and the Psychology of Expectation*. Cambridge: MIT Press.
- Jehan, T. (n.d.). *Max/MSP Externals*. Retrieved 2010, from Tristan Jehan's Home Page: <http://web.media.mit.edu/~tristan/>
- Plans-Casal, D. (2008). Time After Time : Short-circuiting The Emotional Distance Between Algorithm And Human Improvisors. *International Computer Music Conference*. Belfast.
- Plans-Casal, D., & Morelli, D. (2007). Remembering the future: An overview of co-evolution in musical improvisation. *The International Computer Music Association* .
- Robinson, M. (n.d.). *mxj nnLists*. Retrieved 2010, from martinrobinson | audio/interfaces/code: http://www.miajo.co.uk/?MaxMSP:mxj_nnLists
- Rojas, R. (1996). The Backpropagation Algorithm. In *Neural Networks - A Systematic Introduction* (pp. 151-184). Berlin, New York: Springer-Verlag.
- Schnell, N. (n.d.). *ftm*. Retrieved 2010, from FTM&Co: http://ftm.ircam.fr/index.php/Main_Page
- Schwartz, D. (n.d.). *CataRT*. Retrieved 2010, from IMTR: <http://imtr.ircam.fr/imtr/CataRT>