This chapter introduces the interactive installation artwork The Music Creatures *and some of the technologies it provoked — of most note the* coroutine scripting technique *and the* generic radial-basis channel, *which will be extensively explored and used later. It brings the agent-based to a very particular kind of sound-image relationship, and extends both the authorship techniques and rendering techniques of* Loops *and the use of the pose-graph based motor system.*

Chapter 4 — *The Music Creatures*

The Music Creatures are a series of multi-screen, multi-speaker installations that directly investigate the use of the creature metaphor in creating interactive music. Over the course of three years, a number of musical creatures have been constructed and revised. The discussion here will focus on the most recent installation, commissioned by the Ars Electronica festival in 2003, that used an ongoing population of four creatures drawn from a population of eight species.

1. An overview of the artwork

Each creature follows the same design principles: a music creature is one that creates sound solely through the movement of its body, and restructures its body to reflect its current understanding of its sonic world; although networked together these creatures will communicate with each other solely through the air — each creature gets a screen, a speaker and a microphone; each creature lives for a short duration (around 7-10 minutes), with the life-cycle and in particular the learning-cycle of the creatures carefully arranged, by analogy with the sensitive periods found in animals that possess acoustical pattern-learning such

as birds; rather than attempting to capture some complete competence in a particular musical style or cultural context, each creature will have a particular competence in a field that broadly underpins music itself, inspired by the apparently *proto*-musical competences of animals.

This inadequacy of a particular music creature's intelligence to capture a complete musical field or to appropriately capture the complete control surface of its body is a deliberate strategy to create creatures that appear to strive, without long term success, for order and stability. This apparent intentionality, the articulation of effort and the display of the dynamics of expectation and surprise mark the aesthetic goals of this work.

"Bio-musicology" and agent based AI

In this work we are driven by the desire not only to investigate the relationship between musical problems and motor problems but to engage the (necessarily) biological roots of human music and press the field of artificial intelligence into the service of interactive music and digital animation. In this wider context, we have a bold hypothesis: only by beginning with a study of the animal roots of musical behavior — roots which may include the organization of both sound and gesture in time — can we begin to create systems that we can interact with musically. We further hypothesize that traditional, western "high art" music theory might have no place at all in the construction of primitive, artificial, interactive musical agency, and that a study of proto-musical capabilities and the commonalities of animals may ultimately prove more useful for the creation of new interactive musics. Therefore, we make perceptive, learning and motor representations of these music creatures biologically plausible, in the hope that they will be useful for communicative and expressive ends.

For example, the compendium: **N. L. Wallin, M. Björn,** and **S. Brown,** *The Origins of Music.* The MIT Press, Cambridge, MA. 1999.

This often comparative work connects nicely with the ongoing work on the cognitive and neurobiological origins of music. For example, **I. Peretz,** *Brain specialization for music : New evidence from congenital amusia.* In: Biological foundations of music. Annals of the New York Academy of Sciences, 930, pp. 153-165, 2001.

And the tentative relationship between proto-music and proto-language: **R. Jackendoff,** *A comparison of rhythmic structures in music and language.* In **P.Kiparsky** and **G. Youmans** (eds.) *Phonetics and Phonology, Vol. 1. Rhythm and Meter* (pp. 15-44), Academic Press, New York . 1990.

S. E. Trehub, E. G. Schellenberg, D. S. Hill, *Music perception and cognition: A developmental perspective.* In: I. Deliège, and J. A. Sloboda, *Music perception and cognition.* Psychology Press, Sussex, UK. 1997

Despite a recent resurgence of interest in such biomusicology, science cannot yet provide a computationally constructive or artistically useful theory of musical production, consumption or collaboration. And it is unlikely that pure biological approaches will bear fruit without complimentary constructive artistic experimentation.

These musical creatures are early moves toward these goals — they are posed as a response to the problem and draw their inspiration from the problem, rather than as offering any solution to the problem. While their scientific contribution, in terms of either their explanatory or predictive scope, is limited, they do however represent what I believe to be the first artistic contribution to the field of biomusicology — a field that, unlike, for example, artificial life, has so far failed to capture much attention either within the interactive art community or within the computer music community. If biomusicology is to live up to its goals — to revolutionize our understanding of music's relationship with the mind and the human's relationship with music — then digital artists will need to figure out how to play a part.

With regard to animals and music, *The Music Creatures* are a set of agents that:

possess a variety of prototype implementations of **acoustic templates**, in the sense that they segment and understand their acoustic environment (shared with humans) in order to create sound in it. The goal is to create a set of acoustic processes that are capable of simultaneously generating novel material, and of being crafted in ways that the artist considers perceptually intelligible. I hypothesize that to find algorithms and representations which, when placed in the real world, offer the artist a creative balance between surprising novelty and meaningful control, we should look to the algorithms and representations found in nature.

possess abstracted, **simple bodies**, the control of which they learn, and the movement of which create sound. Success will be achieved when there is a certain unity of form: between visual depiction of the creature, the sound that it creates, and its expressive and communicative needs for doing so. This unity is seldom achieved in the plentiful multimedia artworks created to date. I hypothesize that animals provide an ideal example of understandable and engaging expressive agency.

develop on long, environmental, time-scales. They will be works with rather classical form — their narratives will have "beginnings, middles and ends"— rather than a temporal heterogeneity that I believe has become the norm in generative art. Such life-spans of creatures are not without precedent within the agent-based artifact, but the inclusion of biologically motivated and genuine development in such works perhaps is. I hypothesize that by looking at such natural "narratives of development" we can understand how to make interesting autonomous art that unfolds over long periods of time.

We cannot survey all the art that has been, or could be, made with a concern for these characteristics, nor can I cover the variety of possible animal analogues. *The Music Creatures* is simply one extended elaboration on these principles.

Bird song — ontogeny

But before I begin to describe the unfolding of this work, there is much to be gained from discussing a particular natural analogue that was influential in its developments — song birds. *The Music Creatures* are not birds, and unlike, for example *Dobie*, we do not look directly to an animal analogue to find the specific behavior, body and interaction for our creatures. We draw, instead, inspiration from the challenges that song birds face, and the ways in which they face them, to provide a level of "structuring complexity" for the artwork.

This view of the ontogeny of bird song is derived from the long and fascinating literature on the interaction between the innate and the learnt in songbirds. Sources that have been inspirational include:

towards constructible models of song learning — **P. Marler**, *Three models of song learning: evidence from behavior*, J. Neurobiology, 33:501-516. 1997.

an early overview of the problem — **W. H. Thorpe**, *Bird-song; the biology of vocal communication and expression in birds*. Cambridge University Press, Cambridge, UK. 1961.

a later overview **P. Marler**, Song-learning behavior. The interface with neuroethology. Animal Behavior Vol. 30 pp. 479-82, 1991.

and on the connection with human music — **P. Marler**, *Origins of music and speech: insights from animals*. in: **N. L. Wallin, M. Björn**, and **S. Brown**, *The Origins of Music*. The MIT Press, Cambridge, MA. 1999.

Many researchers in classic experiments on oscines have produced a number of well established and fascinating key results, including:

memory based learning. There is a sensitive period for song learning, where infant birds must hear fully developed songs of their species in order to develop normal song. Acoustic isolation, or exposure solely to the song of other species, typically results in abnormal song in adult life. However, birds do not sing until much later than this period (the waiting period in white-crowned sparrows, for example, is around 100 days). This sensory -acquisition phase is generally completely distinct from the motor-production phase. This fact is an unavoidable obstacle to any simple motor theory of song production. *The Music Creatures* make no sound, make little use of their bodies and do not progress normally along their developmental narratives in the absence of sound in the gallery.

subsong. As the motor-production phase begins, birds "babble" — a phenomenon not unlike the one we observe in infants. Amorphous, this babbling doesn't have much connection with what was heard previously. However, birds to seem to require self-feedback at this stage — deafening birds during this period prevents the ultimate production of normal song. It seems possible to conclude that during this phase the bird is constructing sensorimotor mappings for its complex sound production systems. Temporarily deprived of a connection to their motor systems early in their development, the agents presented here would not develop stable musical patterns in later life.

plastic song / overproduction. During the next stage, the song possesses many syllabic elements; some are those heard during the tutoring time and some are "improvisations on a theme". Most will ultimately be discarded as the song crystallizes. Interestingly, analyses of the songs produced in this phase, however, imply that more acoustic material has been memorized by

the birds that than is evidenced by the structure of ultimate adult song. *The Music Creatures* here, through their actions, organize their musical material while playing it, crystallizing a particular pattern or a particular set of sounds.

innate acoustic templates. But in this memorization process birds do not act as acoustic sponges, nor do they soak up all their early sonic environment while they are young. Instead, birds tend to learn only from the songs of conspecifics. Careful experimentation reveals that perceptual and motor limitations are not enough to explain the lack of openness in the learnt song; rather, a certain quantity of innate knowledge about the ultimate song form is required. *The Music Creatures* here sample only parts of the musical domain that they are equipped to understand — one agent looks for rhythmic material, another cares only for timbre. Their predispositions to parts of human music shape what it is that they tend to learn, and what it is that they *can* learn.

While *The Music Creatures* are the basis for no scientific claims, they are clearly not unrelated to natural processes, and they do, I believe, represent one of the first synthetic and artistic deployments of agents that possess some form of biologically inspired development.

2. Narrative descriptions of exchange, network, line and tile

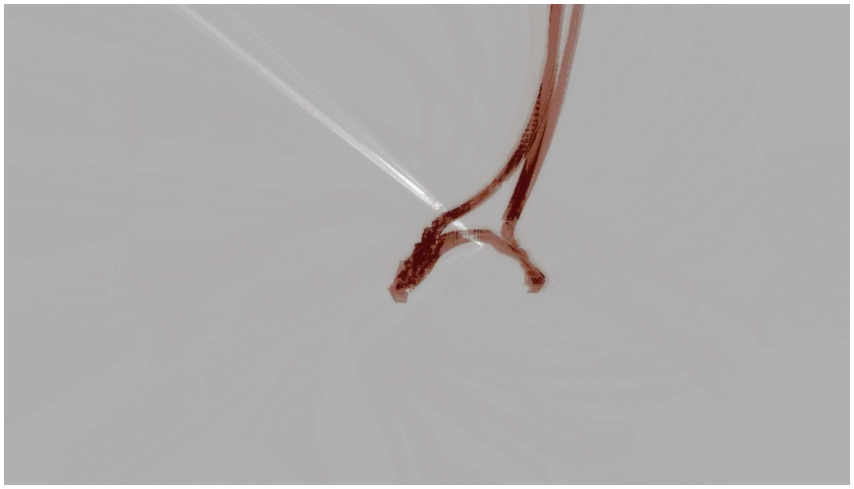
Four creatures from the 2003 version of this work are sufficient to give a sense of the conceptual span of the creatures and the deployment of the technical systems described in this thesis (the remaining four are variations of this set). These creatures are: **exchange** — which constructs and then learns to play a spatial percussion instrument of sorts, while constructing and learning how to move its body; **tile** — a body-visualization of a rhythm analysis that under-

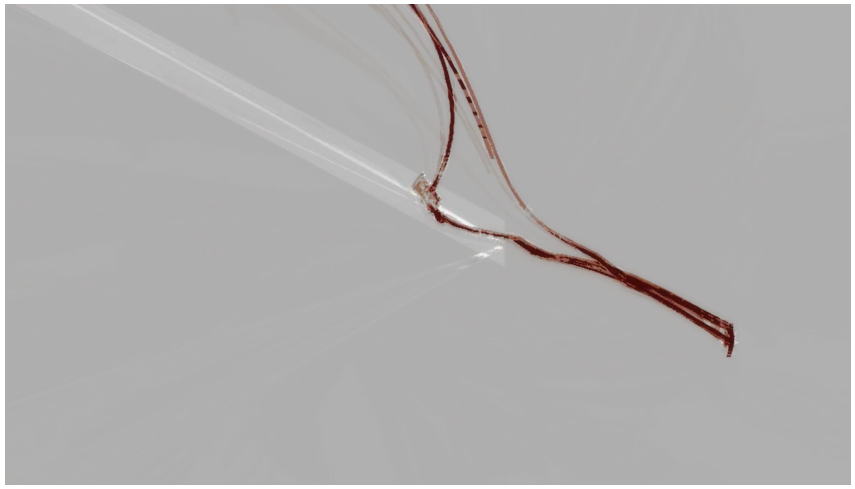
stands and generates rhythmic patterns; **network** — a body-visualization of a simple transition network model of acoustic material that tries using its body to accompany the sounds that it hears; **line** — a metaphorical tape loop that records into an expanding pose-graph motor representation the sounds that it hears and sings.

Each offers a response to the life-cycle of a song bird: they marry both the memory-based aspects of "song learning" with an innate specification for the end of the learning period — and this connection is further underlined in installation where the recording of sonic material is displayed quite visually and sound leaves traces on the screen. Their approach to formulating a sound-image relationship is grounded solely in their bodies — there is no sound without movement and any sonic learning that occurs takes place close to, and is exploited by, their motor-system structures.

Of course, unlike birds their virtual bodies are not necessarily constrained in construction nor in appearance by the physics of the real world. However, *exchange*, *line* and *network* all exist and fight with small "physics" simulations. This underscores in installation the physicality of the sound production that we are undertaking. Neither does a single creature stabilize on a single, robust song for a long period of time — these creatures' lives are evenly distributed between periods of open learning and periods of more closed use. However, this is an approach to the "composition" of music that is analogous to the indirection of birds' learnt song. Rather than specifying the music as notes on a page, the mode of the music's becoming is predefined in the authorship of the creatures. As the work proceeds, the life-cycles of the creatures on the multiple screens diverge, thwarting any direct compositional tendency that remains.

The language used in the following outlines sacrifices detail for the sake of a brief overview. Later sections will fill in these descriptions considerably, clarifying exactly what these creatures do and, perhaps more importantly for this document, how they were made.





exchange

The control parameters for the surface that makes up this creature's body are four rotational degrees of freedom. The surface itself is "skinned" using the conventional skinning algorithm — see page 332.

The exchange creature's body is a simple parametric planar surface embedded in a physics simulation that preserves linear and angular momentum. By changing the control parameters this surface, kinetic energy is injected into the physical system, propelling the creature around its rather small world. Three pre-made "animations" of these parameters, constructed by hand, are played out by the creature, randomly at first as it learns the mapping from surface parameters to the resulting physical movement. This damped, simple physical world offers the opportunity to construct motor-learning in miniature.

In accordance with an emerging general principle in this work, the accumulation of acoustic events are both marked in space and by the indicated growth of the creature's body. This surface, although distorted, is isomorphic to a plane: we place material spatially and graphically (represented as vertical lines) at the most appendage like position, here the fastest moving edge vertex, and growth of new vertices occur around the fastest moving edge. Thus even eventual form of the

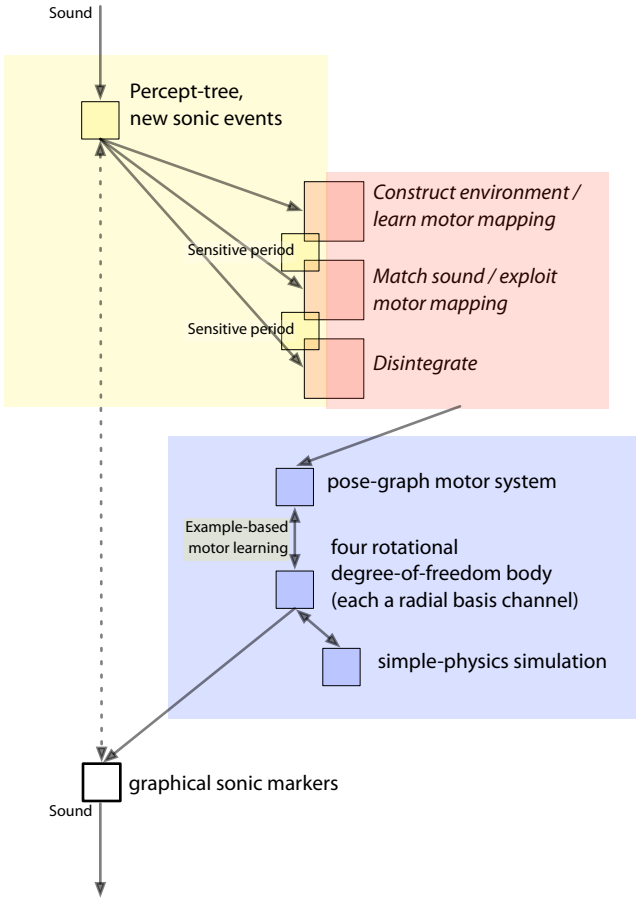
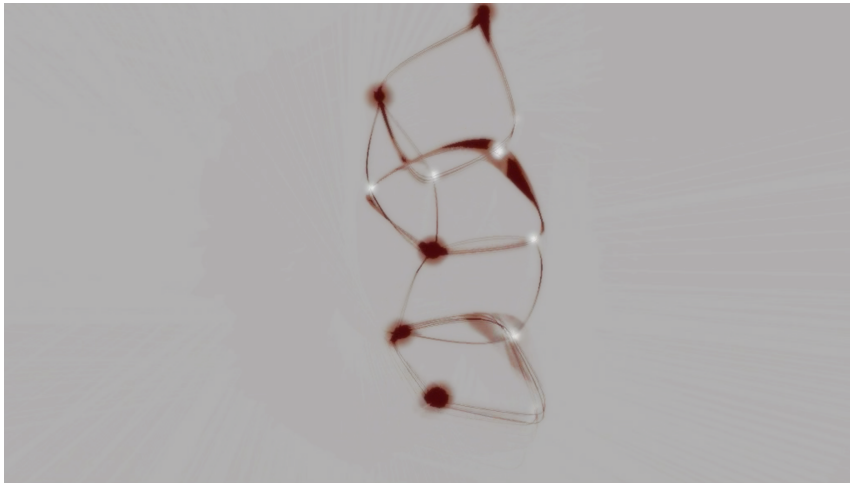


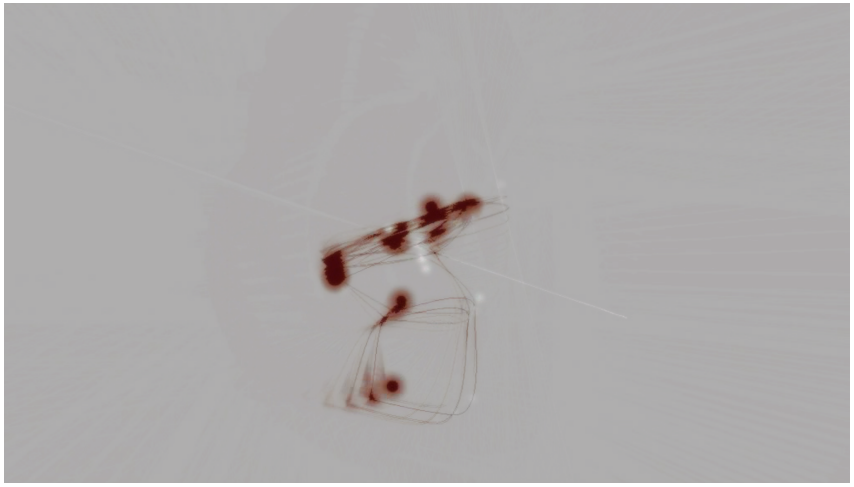
figure 27. An overview of the *exchange* agent.

body is deferred to the agency of the creature and its interactions with the environment.

New acoustic lines distort the locations of previous lines, evening them out. Having accumulated enough body material, enough evenly spaced acoustic material, and a complete enough map of its body control surface, this learning period ends. Having satisfied this 'motor-acoustic template,' the creature now begins to propel itself around the world a little more willfully and a little more flexibly using a more connected version of its pose-graph motor system. Its action system here is still quite simple (and will remain so throughout the piece) moving between seeking notes that are related acoustically with events that it hears, to seeking notes that are as different as possible. The amount of energy that it devotes to moving is coupled to a reservoir filled up with the amount of activity in the room and damped by a longer term increasing fatigue.

Eventually the notes are used up, and as they disappear the body of the creature disintegrates.





tile

There is much precedent for rhythm tracking using phase and frequency locking. In a musical context , the entraining oscillator model is close to the model used in this agent, see: J. D. McAuley, *On the Perception of Time as Phase: Toward an Adaptive-Oscillator Model of Rhythm*. Ph.D. thesis, Indiana University, 1995.

E.W. Weisstein. *Net*. From *MathWorld— A Wolfram Web Resource*. http://mathworld.wolfram.com/Net.html

The tile creature's body is made up of a variable number of flat tiles, and each tile is coupled to a single ongoing model in a rhythm tracking perception system. Each model represents a phase and frequency hypothesis for the acoustic pulses heard in the world. The tiles are connected in a hierarchical structure; initially this structure reflects the parent-child relationship between hypotheses in this percept tree, and free edges for growth are chosen at random. Initially the creature is content to grow its body on a relatively flat surface, appearing like the flattened *planar net* of a complex polyhedron with small oscillations of the tiles governed by the frequency of of the corresponding rhythmic hypothesis.

With enough confidence in the shape of the structure, the amplitude of these

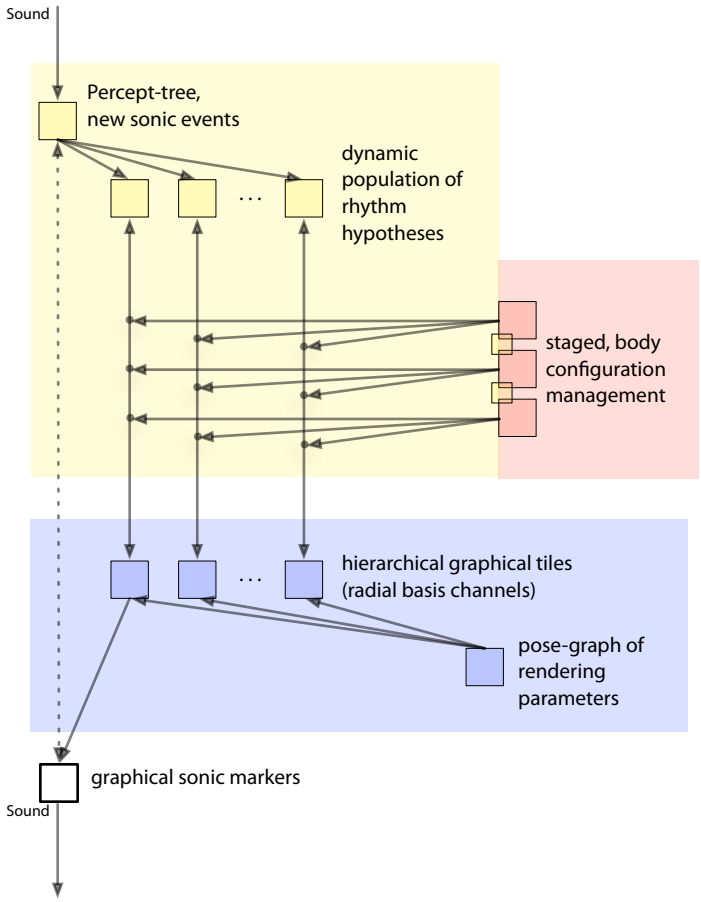


figure 28. An overview of the *tile* agent.

oscillations increases until they become complete rotations. Material that is heard is placed at the location of the tile corresponding to the model that best explains this particular event, and is struck in subsequent rotations. The resulting patterns are strongly poly-rhythmic. Due to the complex folding of this nesting, rotating structure, other tiles can strike these lines — these secondary events are heard as a faint, arrhythmic echo.

The tiles need to be continually reorganized either in terms of confidence (higher confidence tiles migrate to the root of the structure) or, later in the lifecycle of the creatures, frequency (lower frequency models migrate to the root of the structure). During this reorganization it also moves from the complex, randomly branching 3-dimensional structures to what is in essence a single two-dimensional curve made out of squares. After this structure is achieved, subsequent reorganizations drop rather than transfer tiles. The body evaporates.

network

The body of *network* is a point-line connected graph constructed by building a vertex for each identifiable sound event in its environment and an edge between subsequent sound events to indicate temporal succession. It is thus a literal visualization of a transition graph model for musical structure. Duplicate edges are slowly removed and lengths slowly adapted to relax tension in this potentially over-constrained graph structure.

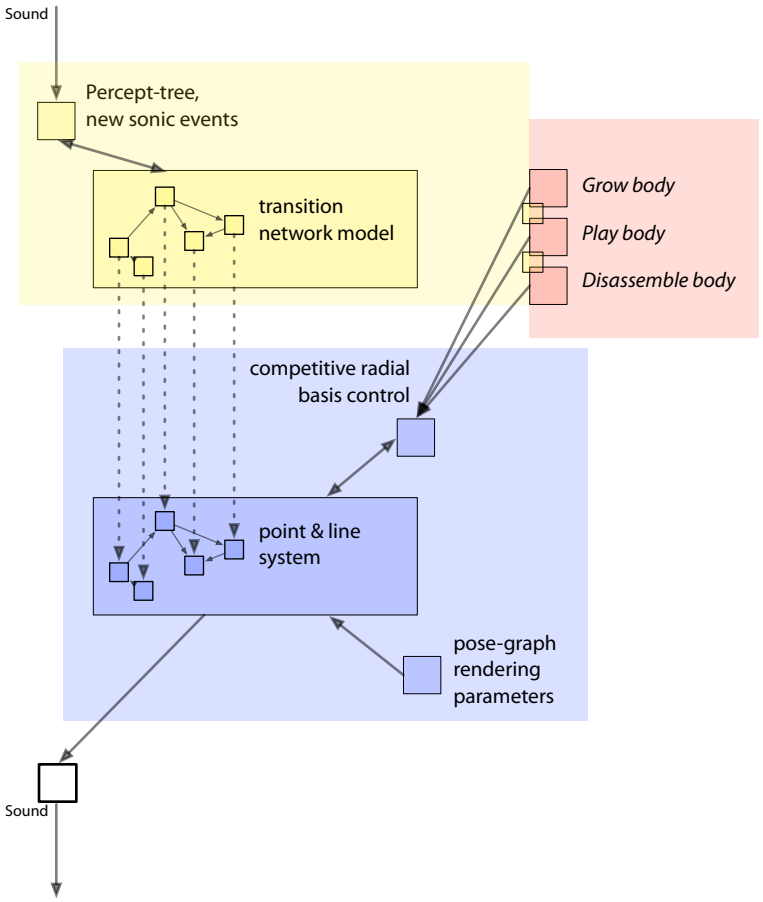


figure 29. An overview of the *network* agent.

After we have a well-ordered graph of sufficient complexity, and after there is a level of confidence in the consistency and relevance of each of the acoustic models, the creature can attempt to respond to acoustic events, by playing the node(s) that are likely to come next given a particular stimulus, after the expected delays. Essentially, *network* "joins in" with musical fragments that it recognizes.

However at any moment a number of hypotheses about the next sound fight for the control of the diagram of possible music that is the creature's body. However, there is a catch, a physical constraint: in order to "pluck" a node, the creature must have physical support from an underlying lattice structure — the nodes either side of a node must be bound lattice before a node can be plucked.

Unheard nodes are eventually forgotten, culled, and replaced by new material. Eventually the culling continues, but this sense of "newness" is not refreshed and the network disintegrates.

line

A single, simple, looped chain attempts to cycle through three hand-drawn shapes — a triangle, a square and a spiral. However, its body representation is an under-specified node representation — local rather than global — so the process of transition proceeds through search rather than as a direct morph or blend. Each node acts through impulsive forces propelled by the sonic events in the room rather than through a direct control structure and thus liable to overshoot, escaping local minima for the search. Faint traces of globally consistent solutions to the problem are indicated through nodes that are getting close to a solution.

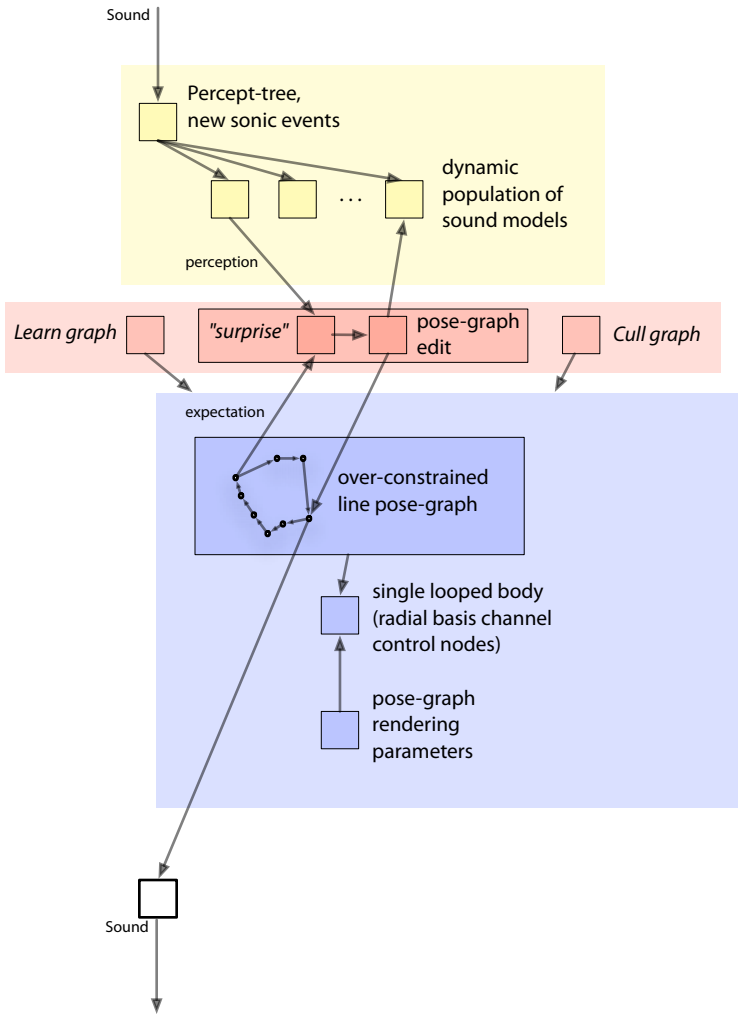


figure 30. An overview of the *line* agent.

The three-state motor sequence is represented in the language of the pose-graph motor system as a looped graph structure. As it attempts to go through the sequence of states, it records the sounds that it hears in those nodes using the pose-graph as a secondary memory structure.

Having successfully toured its graph and learnt the sound of each pose, it begins to sing the contents of the pose as it traverses them, slowly drifting from node to node, but still being propelled by the presence of sound. However, important sonic events that are unlike the contents of the nearest pose-graph node are "unexpected" and cause the insertion of new nodes into the graph, and thus a new pairing: the current state of the body (which has likely been thrown of course by the force of the sonic event) and the sonic material that caused it.

Nodes, especially nodes that have not been successfully visited in a long while are slowly dropped from the graph until only two remain.

3. "Tactical" learning

Eight creatures, the four above and a variant of each, were assembled and tuned in two months, including the time taken to construct the new non-photorealistic renderer that drew their bodies. Despite this short time-frame, *The Music Creatures* departs from *alphaWolf* and *Dobie* in refusing to structure the interaction between participant and agent with an overt narrative or a specially constructed interface. Nor does *The Music Creatures* develop a way of simplifying the world of the agent: rather the creatures are simply given a microphone each.

In this formulation of the agent and world, much is unforeseen. Thus *The Music Creatures* were a challenge — how open could an artwork be to the potential of the sounds in an unknown gallery, their interactions with the agents, and the interactions between the agents?

This time-frame was made possible by the re-use of many of the techniques described in this thesis, in particular the c5 agent toolkit. But specifically, this toolkit had to be surrounded in technologies crafted to deal with both the unforeseen of the interactive and the unforeseen of the authorship process.

We will unpack some of the ones more specifically behind those four descriptions here.

Long-term learning and persistence

The above stories, of course, omit many implementation details; however, the first and most glaring omission are implementation *numbers* — the decisions, thresholds, limits: the choices concerning when and how things should occur. From the above descriptions: "after there is a level of confidence in the consistency in the contents of the node-level acoustical models" — this level is a num-

ber, a **cutoff**; "important sonic events that are unlike the contents of the nearest pose-graph node are unexpected" — this measure of unlikeness is a **scale**, specifically, a mapping from some unspecified range to the domain of 0...1; "Having accumulated enough body material ... this learning period ends" — this is a sensitive **period**, a mapping from some unspecified value scale and a time-period to a decision that some learning epoch should end.

Setting these numbers, or equivalently calibrating the range (and the units) of the quantities that they intersect or scale, is a considerable burden to the author of a complex system. In the particular case of this installation, the burden is impressive: *The Music Creatures* installation was a complex five computer, four screen installation that proved impossible to assemble prior to its arrival in the gallery space — rather the "installation" was made piecemeal, agent-by-agent, in a large open-plan office area. Knowing the dynamics of all four creatures when they begin to hear each other, the precise characteristics of the sound available in the galleries or even just the sound level there in advance was simply impossible.

The impossibility of this kind of foreknowledge of an interactive process is the burden of interactive art. However, it is also the condition and indeed the opportunity of interactive art. To shirk this burden by direct specification is to close off potential before it develops.

However, in each case there is a way of specifying what these numbers should be that is simultaneously more indirect and more explicit. Instead of specifying the number indirectly, we specify directly what this number should achieve and let some simple online learning technique remove this indirection. This, coupled with a rather Lamarkian inheritance of these learnt "traits" across multiple instantiations of the creatures, ameliorates much of the difficulty of setting these numbers and scales. The agents indirectly *learn*, and in learning *specify* the cut-

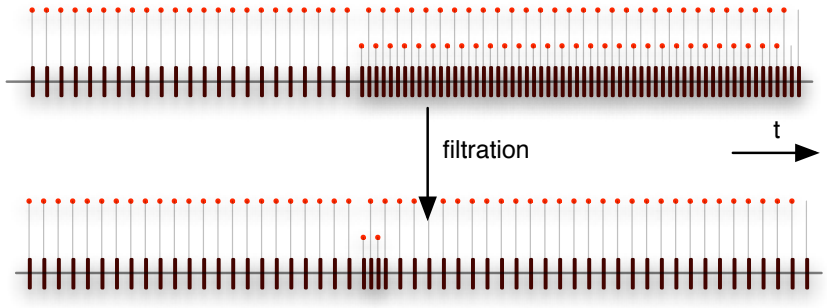


figure 31. An easy cut-off problem. Here we are trying to find a cut-off that maintains the same output rate from a filtered set of valued-events with a non-stationary distribution of values. After a brief moment of uncertainty, the windowed cut-off correctly adapts to a sudden change in the underlying statistics of the channel needing filtering. For much harder examples of this working, see figure 56 on page 253

offs, scalings and period-closure criteria in different ways. We will take each of these in turn.

A **cut-off** tries to form a boundary in range that splits the number of examples on each side into a specific percentage. To find a cut-off, we store the history of examples. A simple estimator would place the cut-off point for an acceptance ratio of α this fraction of the way through a sorted list of example values. This example history is maintained over a very long window of time, since the cutoff itself is rarely changed. Keeping the examples around has the additional advantage that we can change our mind about the cut-off ratio. We shall see this technique, in a more dynamic situation, re-used in the Diagram framework, *page 252*.

For modeling **scalings** we have a richer set of options. Scaling by the maximum and minimum ever seen is one option, but again, the possibility of the data being non-stationary complicates things. The following algorithm uses a simple "effective" maximum and minimum:

initialize min a, and max b, at time $t_a \leftarrow t_b \leftarrow t_0$

then at any time t_1:

$$a_e = a + \gamma \frac{(b-a)}{2}(1 - e^{-\beta(t_1 - t_a)})$$

$$b_e = b + \gamma \frac{(a-b)}{2}(1 - e^{-\beta(t_1 - t_b)})$$

then the scaled version of a datum d is:

$$d' = (d - a_e)/(b_e - a_e)$$

if $d < a$: $a \leftarrow d$, $t_a \leftarrow t$; if $d > b$: $b \leftarrow d$, $t_b \leftarrow t$

this implements a simple forgetting strategy with a rate governed by β and a completeness governed by γ.

For small β and $\gamma \sim 1$ this strategy does not alter the distribution of example values, simply squeezes or stretches it to fit the unit interval; thus many of the properties of the original examples survive. To make the output distribution more uniform, to "whiten" the input distribution, the music creatures use a self-organizing map trained on the history of the data to model the distribution of these data. For scalar input d, for each example presented to the map (in a random order, multiple times) we perform the following iteration:

The classic introduction to self-organizing maps (including the edge-effect phenomenon) is: **T. Kohonen**, *Self-Organizing Maps*. Springer Series in Information Sciences, Vol. 30, Springer, Berlin, Heidelberg, New York, 1995.

over the N "self-organizing" nodes at locations a_n :

$$i = \arg\min_n |d - a_n|$$

for $n = 1 \ldots N$:

$$a_n \leftarrow f(|n-i|)d + [1 - f(|n-i|)]\, a_n$$

where $f(|n-i|)$ is the self-organizing map's kernel function, for this work a Gaussian of radius $N/4$

Additionally, to avoid the "edge effects" of self-organizing maps, we modify the original self-organizing learning strategy: stretching the range of the map on each iteration to be equal to the the effective maximum and minimum computed by the previous scaling formula.

Implementation extensions for this algorithm include: ensuring that $a_{i_0} \neq a_{i_1}$ and using a more robust interpolation strategy (one that takes into account more than two of the self organizing nodes might be to use radial-basis functions after a straight line fit).

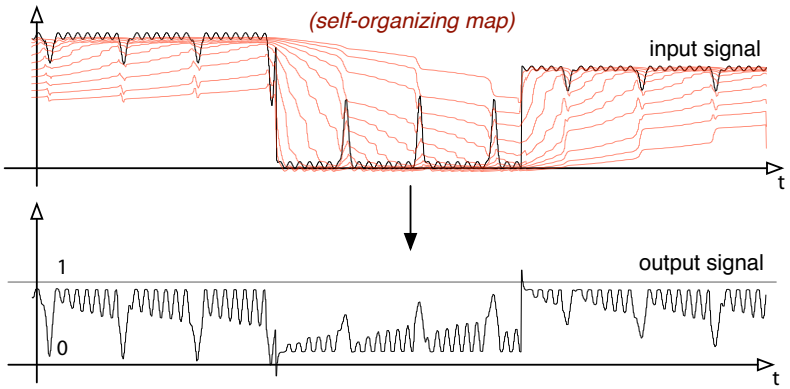


figure 32. The one-dimensional self-organizing map rescales an input signal, "whitening" its distribution, despite large shifts in the input distribution. This makes it an ideal intermediate representation for coupling systems together.

Then, given a new example, we can perform the following lookup and interpolation from this map.

for datum d on map $\{a_n\}$:

$$i_0 = \arg\min_n |d - a_n|$$

$$i_1 = \arg\min_{n \neq i_0} |d - a_n|$$

if $i_0 \leq d \leq i_1$ or $i_1 \leq d \leq i_0$:

$$\alpha = (d - a_{i_0})/(a_{i_1} - a_{i_0})$$

$$d' = (\alpha i_1 + (1-\alpha) i_0)/N$$

else if $d > i_0$:

$$\alpha = (d - a_{i_0})/(a_{i_1} - a_{i_0})$$

$$d' = (i_0 + \alpha)/N$$

else if $d < i_0$:

$$\alpha = (d - a_{i_0})/(a_{i_1} - a_{i_0})$$

$$d' = (i_0 - \alpha)/N$$

Finally, to choose the end of a **sensitive period** based on a value v and a time t is the problem of choosing a cutoff β and a coefficient γ for the following equation:

for a process that yields a value v at time t we pronounce the end of the period if:

$$v + \gamma t > \beta$$

The task here is to accept moments that have high value v but decrease our threshold over time, getting less choosy about the v that will be accepted such that normally the process time t_0. There are, again, a number of ways of doing this, especially if one has information about the process generating v. But in the laziest position —a position of ignorance — by re-scaling v to the unit interval we can eliminate the choice of β and accept if

$$2v - 1 + \gamma t > 0$$

without loss of generality and estimate γ given a set of examples $\{v_i, t_i\}$ and a target time t_0 we can estimate γ from:

$$\hat{\gamma} = \langle 1/t_0 - 2v_i/t_i \rangle_i$$

None of the above "learning" formulae cause any particular theoretical difficulties, especially in the case where a good first guess is available in addition to the indirect specification; for the wide variety of *ad hoc* applications that they are used for in *The Music Creatures*, they all learned and converged quickly. Such techniques are present at all levels of the creatures: at the basis of interpreting sound levels — for segmenting sound into acoustic events; judging when an agent has a sufficient knowledge of its sound-body mappings — so that periods can end; comparing sound models — when is a sound "new", when is it surprising? It is inconceivable that this piece could have been completed without the tactical widespread deployment of these simple techniques; it is similarly inconceivable that such a piece would have been attempted without them.

The methodologies of learning

However, such technical competence is not the end of the story. There is a significant gap between the academic reality of this simple learning and its practical use, its integration into a working practice. This "implementation difficulty"

stems from the way that agents are authored, indeed it stems from the fact that the agents are *being authored* while this learning is going on. This makes how this storage and recall of this data occurs surprisingly tricky.

Ideally, we'd like to specify our numbers, cutoffs, scaling etc. with as little fanfare as possible:

```
PersistedScaling magicNumber = persistedScaling("cutoff-for-loudness", 0.0f, 1f);
```

this declaration loads a long term (that is, longer than the execution life of a creature) model, initializes a model if there isn't one already (to be between 0 and 1 in this case), prepares it to collect examples upon use:

for cutoffs: `passed = magicNumber.filter(value);`

for scalings: `value = magicNumber.filter(value);`

for periods: `passed = magicNumber.filter(value, time);`

and arranges for it to be saved upon the termination of the program.

Clearly this persistent data faces the same challenges as faced by the persisted rendering and action attributes of *Loops, page 98*. Part of our defense against the system changing from invocation to invocation is our handling of non-stationary example sets.

But there is an additional problem not faced by the *Loops* database, an additional dimension to the problem of "naming" in complex systems. The issue stems from the need to exploit as much context and data as possible for new learnt parameters — as much context and data, that is, and no more.

Consider that multiple creatures may execute the same lines of code as above; consider that the same creature may instantiate many objects that have that line in them; and finally, consider that we might have a small test program that thoroughly tests this line of code before it is embedded in a larger agent. This line appears as a simple declaration, but it is woven into my creative practice in an intricate fashion. In each of these cases, we might wish to exploit the knowledge accreted in the test programs, sharing knowledge accumulated between invocations of related creatures. Thus, we wish to structure our persistent parameters in such a way that as authors we remain mobile, and that the data can move with us: that we can move from small test stage, an exploratory phase, testing inside a larger system, back to a small debug session.

A context-tree based solution *page, 195* can help with the problem of multiple instantiated learnt parameters — a parameter instantiated in a different context counts as a separate, different learnt parameter that might share past history with parent contexts. Since the current context is an unambiguous chain of names it can be stored just like any uniform resource locator. However, this context does not suffice as the sole context for these parameters.

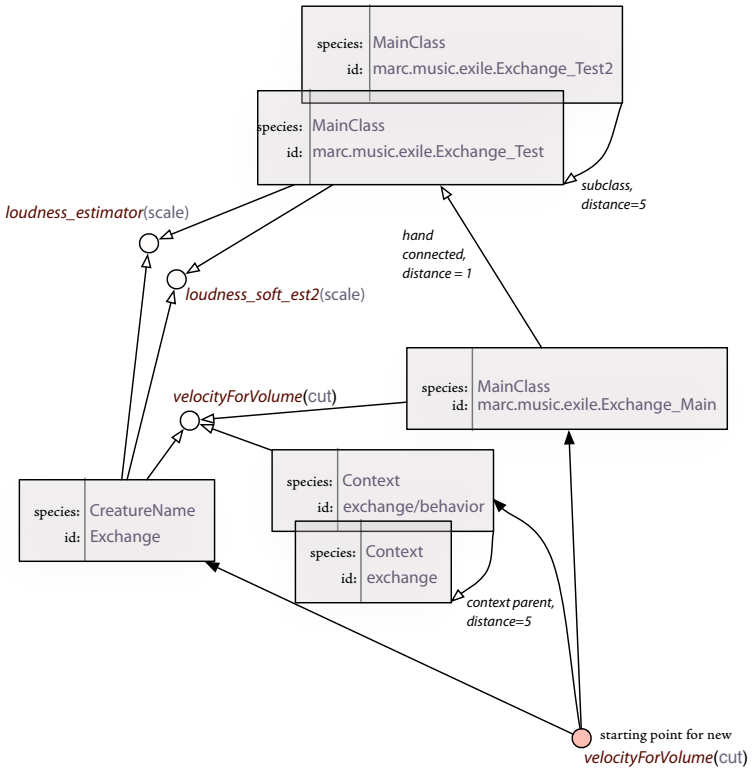


figure 33. Initialization material for newly created models is found by breadth-first search across the database. Here a new velocityForVolume(cut) would be initialized with material from a previously learnt model, learnt in the same main class (as well as the same context, and the same creature name). This node is then placed in this graph at this location.

We can list a number of potentially important "contexts": the "name" of the creature, or relatedly, the name of the "main class" (the entry point into the entire program running the agent), and finally the name of the learnt parameter itself. Unfortunately, there seems to be no single ordering of these contexts that makes sense.

Rather than attempting to formulate a single tree structure with these learnt parameters at various leafs and nodes, we form a heterogeneous, but directed, graph with contexts and learnt parameters connected together. Collecting information about the data that should be behind a learnt parameter is done by breadth-first search over this graph structure.

Many of the edges of this graph can be established automatically — for example edges are formed from a main class to its superclass (but not the other way), if the superclass existed in the graph. Other weighted edges can be created as well as permanently deleted with a graphical utility (for later work, this was assembled and written in *Fluid*). This same utility rolls back the database to a previous state (when we make an experimental change); tags a database state with a useful label (something more easily remembered than a date and something that tracks the state of the project); and can list differences between database states (to allow an investigation of the project as it is changing).

This connected-graph-aware versioning system is an extension of the ideas used for *Loops*'s rendering parameter database. It is thus neither the first, nor will it be the last persistent store required, *page 209*; required, that is, to take a technology that in an academic sense is working perfectly, and turn it into a technique that actually integrated into a working practice.

4. Advanced flow control — coroutines, radial-basis channels, and an introduction to the "language interventions"

The previous section identified a particular class of unobtainable foreknowledge in authoring interactive works — knowledge about both the coarse structuring and the fine details of the world that the work finds itself installed in. In finding a few algorithms, it built the necessary support to ensure that they could and would be widely deployed throughout the work.

Another broad class of uncertainties faced by the agent author is less concerned with the world and the agent and more concerned with the interactions between systems internal to the agent itself. Specifically, the uncertainties faced where multiple parts of an agent fight for, or cooperate over, the control over another. Such arbitration and control problems occur throughout the agent metaphor. Some are action-selection problems — objects fight for attention, action-tuples compete for expression. But others are coupled more directly to perceptual and motor systems.

Imperative programming languages earn their name by their focus on a sequence ordering of commands. With a small number of exceptional flow control structures, programs in such languages start at the top of the page, and execute line-by-line until they get to the bottom. Advanced metaphors, constructed "on top" of imperative languages, increasingly complicate this flow.

The c5 toolkit makes extensive use of one set of such complications — object orientation. However, as we shall see in a number of places in this work, there is an essential conflict between the flow of serial execution in an imperative programming language and the flow of time in the life-cycle of an agent. These conflicts necessitate a less generic set of extensions to the set of non-imperative flow control structures, ones specifically tailored to ameliorate the conflict be-

tween the orderly imperative of what it that an agent author is trying to achieve this moment and the disorderly, diffuse chaos of other parts of the agent framework already assembled.

As we proceed down the perception → action selection → motor control chain, this conflict becomes increasingly severe. Although the perception system monitors the world as it finds it, in realtime, for the most part its flow control can be coupled quite directly to the world and only occasionally reconfigured by an action system — the world dictates what code gets executed when. As we move to the parts of an agent that take responsibility for maintaining long-term behavioral coherence do the differences between the flow of this coherence and the flow of an imperative language begin widens to a chasm.

In this work, and many other agent systems, by the time we have reached the action-selection layer we have small parcels of imperative-code (the action payloads of our action-tuples) and a highly distributed flow-control structure (the action selection mechanism and its coupling to other systems) that changes often and changes on different time-scales. None of this code is written in what one might call an imperative spirit. As we head towards the motor system, even these parcels of code need to be dissolved as the granularity of control becomes equal to the granularity of the agent's "update cycle", essentially the frame-rate, the control-rate, or the clock-quantum of the simulated world. Programming at this level is hard — hard to write, hard to maintain — a barren landscape for artistic intention because no thought, it seems, can be continued in any meaningful way for more than a few lines of code.

A good introduction to continuation-passing-style that is less language specific than many is: **D.P. Friedman, M. Wand, C.T. Hayes,** *Essentials of Programming Languages*. MIT Press, 2001.

The canonical definition of the Java language is **J. Gosling, B. Joy, G. L. Steele Jr., G. Bracha,** *The Java Language Specification, 3rd edition*, Addison-Wesley, 2005.

for Scheme, the following is useful: **G. L. Steel, G. J. Sussman,** *The Revised Report on SCHEME*, MIT AI Lab Memo 452. 198.

for Python, see http://www.python.org

for the Common Lisp Object System, **G. L Steele,** *Common LISP: The Language*, Bedford: Digital Press, 1990.

for C#, http://www.ecma-international.org/publications/standards/Ecma-334.htm

Part of the response to the problem will be to look to more sophisticated flow structures. I'll use this as a basis for a critique of current digital art-making environments (and the concepts they encourage). And I shall show our main, conventional object-oriented programming languages hybridized with ideas from other languages. When this hybridization occurs, it will concern flow control features: I will introduce a "continuation-passing style" implementation in Java — inspired by languages such as Scheme; a "coroutine" based system implemented in both Java and Python — inspired by Python's implementation, but finding its roots from the days before ubiquitous threading; and a set of "complex, multiple dispatch" techniques inspired by the Common Lisp Object System.

Adding these exotic language extensions and libraries to a base language requires some justification. The choice here — Java — is a popular language that has found and thrived in a fertile niche between sophisticated abstraction (the reflective capabilities of the object model and the malleability of the virtual machine code) and a practical rather than polemic level of object orientation tried and tested in other languages and implemented well. It offered support for exceptions, a safe memory model and a runtime type system at a time when implementations of these features in other mainstream languages were experimental and poorly tested. It seems a suitable language for large, complex collaborative development especially in teams with a variety of skill levels and today is surrounded by a number of sophisticated programming environments.

Where it falls down is perhaps on the smaller scale, and it is here that the so-called dynamic languages take over. Removing the type system and its textural burdens (declarations, casts, and various commitments to organizing one's abstractions), these languages are generally considered ideal for testing, for gluing together pre-existing objects and for "interstitial coding". They have a tendency to encourage language features that are thought by many to thoroughly scupper the maintainability of large, multi-person code-bases. At this price they often

offer direct or indirect support for malleable syntax and modifiable meta-class semantics that allow the appearance and benefits of domain-specific languages without any of the burden of writing and maintain one's own tool chain. As such we augment the choice of Java here as the main programming language with a more dynamic language — the Java-based implementation of Python. It's important to note that its role as an *interstitial* language in this work is enhanced by the open-source nature of this implementation, and that it, rather than Java, forms the basis of the programming environment developed in the last section of this thesis, *Fluid*.

The choice of language for this work is not casual, indeed part of the argument of this thesis is that programming languages and the tools around them are so malleable, interesting and important in one's work that the artist bears a responsibility for the choice of language and the design of the language — that the artist should not be a passive consumer of a toolset. And while discussion concerning the relative merits of programming languages typically generate more heat than light, the implementation details of the broad concepts and the language level interventions of this thesis remain significant even given the perennially unjustifiable choice of language itself.

In similar languages (and there is a lot of interest in Java-like languages — for example the more recent C#, or perhaps even the contemporary style C++), it is expected that some of the implementation techniques, or at the very least styles, will carry over. In other different but similarly sophisticated languages, these techniques will no doubt be achievable in some other ways, and the implementation specifics here will have little to offer. Still, the arguments for their applicability to the problems discussed here, extension to the programmer's palette and place in agent architectures in general, are expected to stand.

This "imperative disconnect" is the second thing that the narrative descriptions of the music creature's behaviors obscure, or at the very least gloss over. How is the passing of time authored? — what happens in the space between paragraphs? Quoting again from the descriptions: "Having accumulated enough body material ... this learning period ends" — this is a *scripted* transition that waits for some condition to be true and takes one action system configuration to another; "nodes ... are slowly dropped from the graph until only two remain" — this is another *scripted* condition; "in order to 'pluck' ... it takes the nodes either side ... binds them temporarily" — this is a *scripted* motor program.

We have a number of techniques, not least of all the action systems themselves, for waiting for a condition to occur before doing something. But perhaps there ought to be a lighter-weight, and more imperative, way of specifying these well ordered sequences of tasks. Even the "payloads" of actions themselves often turn out to have a "scriptable" life-cycle — "do something when at the start, keep doing this for a while, and then make sure that this is done before finishing". These tendencies are not confined to *The Music Creatures*. For example, from the action-tuples of *alphaWolf*: to set this variable in working memory, tell the motor system to perform a particular action, wait until it is done, or for a while; go over to that other wolf and *then* growl. These are scripts in miniature.

There is a role for such light-weight languages in the cases where the order of events are predicable and the number of possible paths and interactions through the script code are small. We see such small programs persistently at the motor level — for example: action systems ask for a particular goal to achieved and the motor system must navigate to that task by calling up a walking animation until the agent is close enough to take a half-step and then, shortly after, it finally can sit down. This is a suitable problem for scripting (as opposed to a finer grain

representation like a perception / action system) because the order of events is well known and the behavior in the case of interruption is well defined — for such simple things there should be a way of just writing them down and having them execute. The remainder of this section is devoted to a technical description of the incorporation into Java of a language pattern that allows just this.

We have seen repeatedly that sequencing action over multiple execution cycles usually requires special effort for imperative programming languages. Typically, either effort is applied to rethink the sequencing in a way that is *reentrant* (in essence using a repeatedly executed switch statement) or to build the sequencing by imperatively constructing a kind of executable data-structure. We can do better using the reflexive powers of a language such as Java to implement an older programming idea called coroutines. A coroutine is a restartable procedure (or here, method call). Coroutines are easily implemented in a language with proper continuations, or even reflexive access to the stack-frame structure and they look to be implementable in Java with some load-time byte-code manipulation — so the implementation comments here are of less interest to other language implementations, but how coroutines are used should still be interesting. Now we can make apparently imperative scripts →

for precedent for these"continuations" in Java, the RIFE project: http://rifers.org/wiki/display/RIFE/Home

The method used here relies on the the ability to recover the order of method declarations in a source file at execution time. The order is compiler-dependent; however all compilers used for this work since 2000 (both as part of the standard Java developer distribution and as part of the Eclipse project) have emitted byte code that is ordered in this way.

The Eclipse project: http://www.eclipse.org

```
rc beginning(){
        System.out.println(" beginning ");
        return progress;
}

int i = 0;
rc waitAWhile(){
        if (i++<10) return wait;
        else return progress;
}

rc thatsIt(){
        System.out.println(" that's it ");
        return stop;
}
```

This object is given to an executor that interprets the return codes of these methods. The critical point here is that at each return the rest of the system (and in most cases this is almost all of the rest of the agent) gets a chance to execute. Although for the purposes of the above example we would certainly be better off in a language that supported coroutines directly (for example, the above is one method and six lines in Python), the fact that we are forced to return something that describes the next move (here: wait, proceed or stop) turns out to be an interesting opportunity. For, to be powerful enough to be up to the tasks that we ask of them, our scripts (clean as they are) need to retain some of the best features of object-oriented programming. In particular, it would be better if they were a little more reusable, a little more component oriented and more composable.

To do this we begin by extending the lexicon of things that our coroutine method elements can return and making the executor keep track of a stack of coroutines, only the top of which is executed. This is, in a sense, a recreation of

the stack behavior of the virtual machine, with added reflexivity and flexibility — a kind of homemade *continuation-passing style* of programming. In addition to the singleton objects, ***progress***, ***wait*** and ***stop***, we add: the singleton ***failure*** — which signals an exceptional "error" condition has occurred; **push**(*coroutine*) — which pushes a coroutine ahead of this one on the stack; **replace**(*coroutine*) — which replaces the current coroutine with another; **after**(*coroutine*) — adds this coroutine to the stack just before the current one (it will thus execute after this one); **trap**(*coroutine*) — upon failure, the default behavior is to tear the stack down and return failure; adding a trap to the stack intervenes in this destruction, and allows the trap to execute instead.

From these primitives we can build more useful coroutines that exploit the stack structure for their implementation but don't necessarily reference it in their definition. For example: **andWhile**(*coroutine[]*) — while these coroutines are progress-ing or wait-ing, keep running this coroutine, otherwise stop; **needs**(*coroutine[]*) — needs all of these coroutines to progress before this coroutine progresses; **obtain**(*coroutine[]*) — a combination of both of these, wait until each coroutine progresses before continuing and only run while they aren't stop or failure.

The name of the last "return code" hints at where some of this is heading. Coroutines provide an excellent interface to a resource model. The simplest resource is something that only one object can own. Other objects need to wait in line to own the object or can a perhaps steal the resource, forcing the lock. If they are waiting in line, then their coroutine-based attempts to own the resource are *wait*-ing too, if they force the lock, the coroutine based interface used by the ex-owner returns failure. Through obtain and needs we can attempt to claim a number of locks at the same time and give up if no progress is made on any front and try something else.

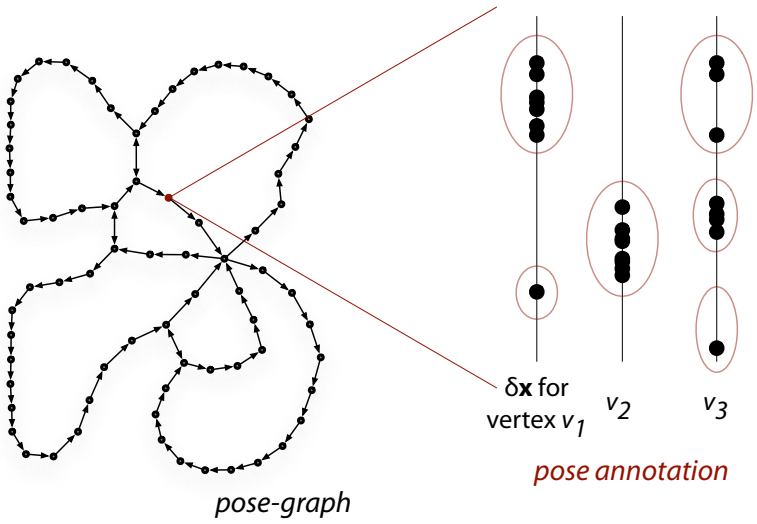


figure 34. The motor learning in *exchange* is performed by annotating the contents of its primitive pose-graph with the effect of performing those poses. Theses annotations store the effective directional change of three of the bodies vertices. Given this structure the pose-graph can now be searched in a different-way, based on these annotations. While not accounting for the momentum that the creature has, this is sufficient to allow a certain, perceptible, amount of goal directed movement by the creature.

These coroutine / resource combinations are extremely useful for guarding access to limited resources throughout *The Music Creatures* and successive work. They are used in a strictly technical way — apportioning use of the two running video texture channels in 22 and making sure that these textures are switched out only when they are not on the screen; and they are used in a more "agent oriented" way: distributing the motor-program control over the reorganization of *tile*'s body segments — a reorganization which is a short series of uninterruptible, but potentially overlapping, procedural animations; forcing the "physical constraint" that *network* suffers under before it gets to play its nodes; and of course, limiting the life-cycle transitions out of sensitive periods (essentially, executed with needs({*aCertainAmountOfLearning*})).

The coroutine / stack model also generalizes the previous stack based model for motor programs, discussed *page 65* that had surprising longevity — it was used for *alphaWolf*, *Dobie*, *Loops*, *Max*, an early version of *The Music Creatures* and still makes an appearance in current work at the MIT Media Lab (for example, the work of The Robotic Life group). This simpler model aimed first for the reusability of components — stacks were assembled of motor programs that were themselves made out of stacks of pre-made components — but what was lost was the scriptability. Nothing much was left of the underlying imperative language (Java) which, of course, has much in its favor when it come to the task of programming. Further, it had no equivalent of trap(*coroutine*), and no exceptional error handling which made conflict handling and disassembly of motor program stacks troubling and error-prone.

for an explicit discussion of Leo and the pose-graph: **C. Breazeal, D. Buchsbaum, J. Gray, D. Gatenby, B. Blumberg,** *Learning From and About Others: Towards Using Imitation to Bootstrap the Social Understanding of Others by Robots.* Artificial Life, 11 (1), 2005.

I am indebted to Robotic Life group researcher **Jesse Gray** for bringing these ongoing issues with the single stack model to my attention, long after I had abandoned them myself.

Blumberg's motor system is described in: **B. M. Blumberg,** Old Tricks, New Dogs: Ethology and Interactive Creatures. Media Laboratory. PhD Thesis. MIT. 1996.

Minsky's centrality of (negotiable) resources is in his forthcoming *Emotion Machine.*

These extensions are not simply academic investigations into exotic programming models — the lack of any formal cooperation between motor system stacks was only tenable for characters who had only one primary motor system to begin with, and characters, such as the Robotic Life Group's *Leo*, that assemble many layering motor systems together have spent much time constructing such cooperation. Its lack of a fine-grained resource-like view onto the motor system and its insistence on a single stack bought some simplicity at the expense of much power.

While the formal principles of this "new" language for motor programming are new, the underlying insight that there is a lock/resource model aspect to motor systems is hardly novel. The gated access to shared resources forms a core part of every modern operating system for as least as long as multitasking; the importance of necessary and preferable locks forms one of the key insights of Blumberg's motor system. In a broader context still, Minsky places resources with apparently similar life-cycle constraints at the core of his recent AI architecture — but the *scriptability* of the life-cycles of these resources in an agent framework is what makes them powerful in my formulation here.

Adapting bodies

The Music Creatures is as much about the bodies of the creatures as about the music that they make, and more to do with rethinking the figurative in computer graphics than about reinventing a fragmentary minimalism in computer music. The third thing missing in the narrative descriptions of their behaviors was the specifics of the control structures the agents had over their bodies. Again, quoting from the narrative overviews:"...its body representation is underspecified and node local" — this is a specific kind of body representation that can be used to generate movement; "the creature must have physical support from an underlying lattice structure, it takes the nodes either side of the node to

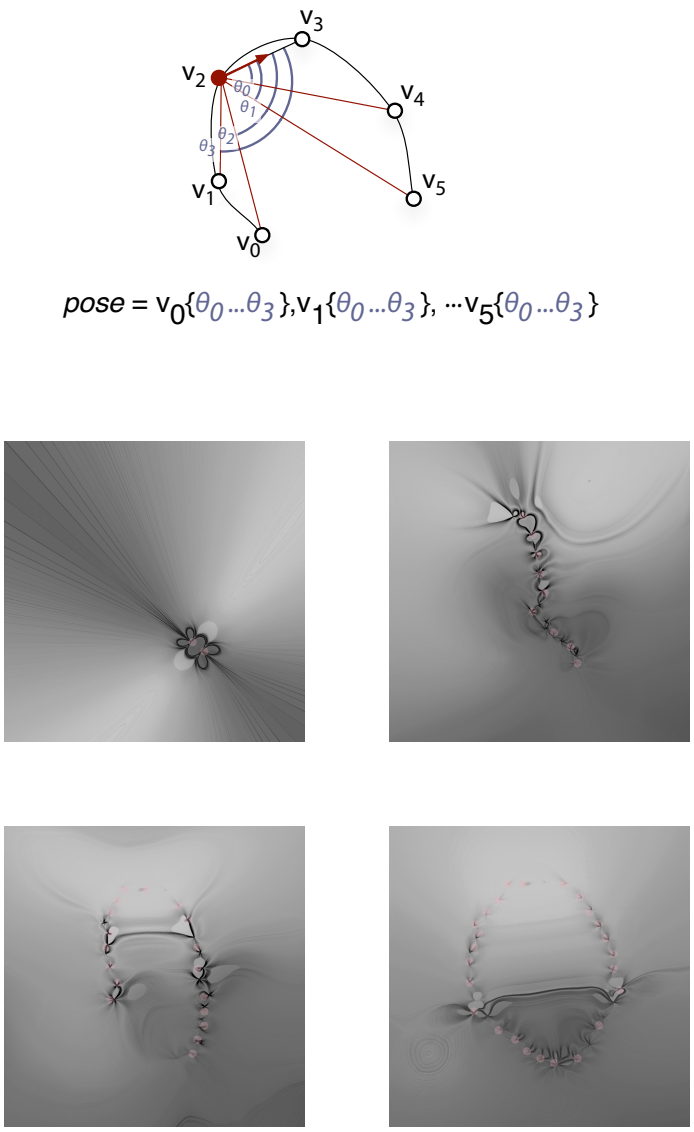


figure 35. *Line* has a deliberately perverse pose representation. Rather than remembering the location of each of its control nodes, the pose records an angular representation of where each node is with respect to each other. *Line* attempts to move from pose to pose using this representation embedded in a spring-like physics system. The result is a pose representation that is capable of transitioning between material, but does with effort and difficulty. This pose representation also gives rise to the potential fields illustrated in the lower half of this figure. These fields show, given a fragment of the growing line, where the next control node should be placed in order, here, to create a circle.

play and binds them" — this is procedurally generated animation of an altogether more flexible kind of body. "... as exchange learns the mapping from surface parameters to the resulting physical movement" — this is learning to procedurally control a body.

The pose-graph, although flexible, isn't for every situation. It handles very well the case when the control over a body can be discretized, even if those discrete units are small blend spaces. It isn't appropriate for bodies that offer a high number of independent degrees of freedom that are not broken down, or do not need to be broken down into "animations" or "poses". *Network* and *tile* are creatures with such a body.

Before moving onto the more abstract control problems that the music creatures face, we should look at how they do exploit the pose-graph motor system to support the simple learning that they achieve.

Both *exchange* and *line* incorporate learning into their use of the pose-graph — both are simple, data-driven, example-based techniques that gain any power they have over the body of the creature because of the framework in which they are placed.

The problem faced by *exchange* is to remember what effect its pre-made animation material has on the navigation of its body through its world. It will then use this knowledge to navigate back to various locations in order to "strike" the musical material stored there. It accomplishes this simple task by annotating its pose-graph — which begins with three, simply connected "animations" that exploit the creatures four rotational, parametric degrees of freedom — with the results of playing out that frame of animation. The multiple examples remembered from the ultimate translation vectors that result on the creatures' corner vertices are stored in a clustering structure not unlike that of *Loops*'s rendering

parameters. Having filled this data-structure with examples, *exchange* can search outwards through its motion graph in order to synthesize movement that propels the creature toward a particular point in space. The creature itself is "damped" back to the origin of its world, so it can never go too far astray.

Line operates in a similar way, by annotating the pose-graph structure. Here, however, it is not the consequences of movement that are stored in the nodes, but a record of the sound present when the pose has been played out. Perceptual clusters of sound that fission, propagate topological changes to the pose-graph itself — new pose-material is sampled directly from physics-based body.

Finally, it is important to revisit our general description, *page 69*, of the role of the motor system in an interactive work in the light of these simple learning techniques. We have already seen the motor system's primary force in organizing pre-made material, incorporating "content", made using other, non-agent techniques into the agent and becoming the location where this material is blended, spliced, resequenced. With the simple learning techniques described above, the surface between the agent and the pre-made moves outwards, in favor of the artist. The agent can now use material, not as "content", but as scaffolding for learning, the seed for agents own material.

The generic radial-basis channel 1 — flow blending

In general, however, the pose-graph has little to say about the problem of the body-representations for non-figurative creatures — it offers a control structure for the body for *line* but not the structure for it to control. It will have even less to say for the body (as opposed to the pose) representations for many of the creatures we see later that are adapting their bodies or sampling them from motion-capture data. One generic framework for the construction of these bod-

ies will be presented soon, *page 300*. But we can treat some of these concerns here.

To review: there appear to be two essential and unavoidable characteristics of the environment that "motor-systems" work in, two unavoidable "unforeseeable" problems in authoring agents: firstly, multiple pieces of code (for example, action-tuples) attempt to access the same resources (for example, joint-angles) and these read and write accesses must be arbitrated, blended, or rejected outright. Secondly, these processes want to have their accesses at one particular time-scale, duration or rhythm (for example, when they become active) but want their effects and control to last over a different time-scale (for example, starting slowly, but continuing until a goal is achieved or until a goal is unreachable, fading out over time, etc.). We'll call the first of these the **blending / arbitration problem** and the second, because it comes from a disconnect between the flow of execution in our action systems and the flow of execution in the motor system, the **motor flow control problem**.

Both of these problems are approached by the coroutine / resource model given above, but only in the case where the arbitration and the flow problem is in some way *localizable* and only when the control itself is *symbolic*. This is often the case in high level terms in pose-graph motor systems, where there is a well defined surface between the action and motor system — and the currency of exchange across this membrane are named (that is, symbolic) poses.

However, at the lowest levels of the motor system, the communication with the body, there are often *blending* problems that cut across coroutines and are numeric rather than symbolic. These numerical blends are often the kinds of control that we wish to except over processes — when the exit of one routine should be *smoothly* eased-out, and the entrance of another should be strongly percussive. As the motor systems that are constructed for our agents become

progressively more hybrid — incorporating both "content driven" pose-graphs and "process-driven" procedurally generated movement, these blending and layering problems will dominate as a whole host of distributed processes throw material at the control structures of the agents' bodies.

We will start by discussing a simple but powerful general-purpose approach to constructing motor-system elements that address these two problems — the generic radial-basis channel. This has worked well for the creation of some parts of not just *The Music Creatures*, but every work since.

In its simplest form, a channel is a composite of the following elements: a value representation; a number of "postings" to the channel; and a (monotonically increasing) time-base. A value representation exposes a mathematical space that is at least as large as a vector space (we shall note in passing the possibilities of value representations that are non-commutative and thus are not vector spaces or even fields) in terms of the following primitive operations:


```
interface ValueRepresentation<t_value>{

    t_value zero();
    t_value add(t_value a, float w, t_value b);
}
```

These two operations are chosen to be the minimal set required to implement an incremental weighted average. A posting is an object that has the ability to compute, given a time-base, a *value* v that is compatible with this value representation, a scalar *weight* w and a *time marker* T. Given its set of postings and a value representation, a channel can compute its value at the time indicated by its time-base by summing all the weighted values of its postings. In addition to computing its value at each update cycle, the channel also considers culling postings from the active set: postings are removed if their net relative contribution to the channel falls below $\varepsilon \sim 10^{-6}$ and the time-base is after the posting's time

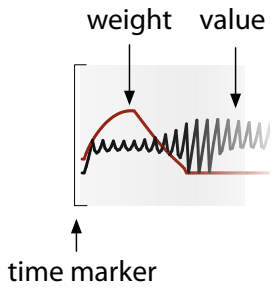


figure 36. The anatomy of a "posting" to a generic radial-basis channel — a weight function, a value (both of which vary over time) and a time marker (which may shift after posting).

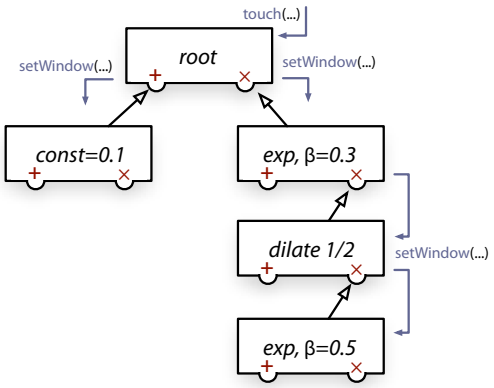


figure 37. A library of generic window functions can be created and combined using this "plus / multiply" tree structure. Each of these windows can be dynamically expanded in response to setWindow(...) messages.

marker. This arrangement shares responsibility for the removal of a posting, ensuring that elements that have an important effect on the value of the channel are kept and that postings that temporarily have little effect can prevent themselves from being removed (by moving their time-marker into the future).

The implementation of a posting can generally be broken down into a window generator, which generates the weight for the posting, and a value generator, which decides on the value. Critically, the the value of a channel is changed in a very orderly fashion — first each posting is updated (in addition order), then each is asked for a value and a weight, then these values and weights are combined incrementally using the value representation, and finally, the channel checks for culling possibilities. This allows postings to immediately remove themselves from the channel — by setting their weights to zero, and their time-markers to $-\infty$ — in the knowledge that they will never be updated again. Further, this gives new postings access to the value and total weight of the channel prior to their entry — useful for smoothing their entry and setting the scale of their windows.

With these simple features it is easy to build a lexicon of window functions and generic value generators. Window functions can be multiplied, blended and added for they are just scalar functions and the value representation interface is complete enough to enable the creation of various filters without binding to the underlying representation. Window functions typically provide compact support in time — hence the name of this over-arching framework: generic *radial-basis* channel — and also share some of the control over the time marker calculation. Both these components are reusable and easy to write. But it is the control over the "posting" and its life-cycle, the relationship to the post-*ing* to the post-*er* that is more interesting and what took us to this structure in the first place.

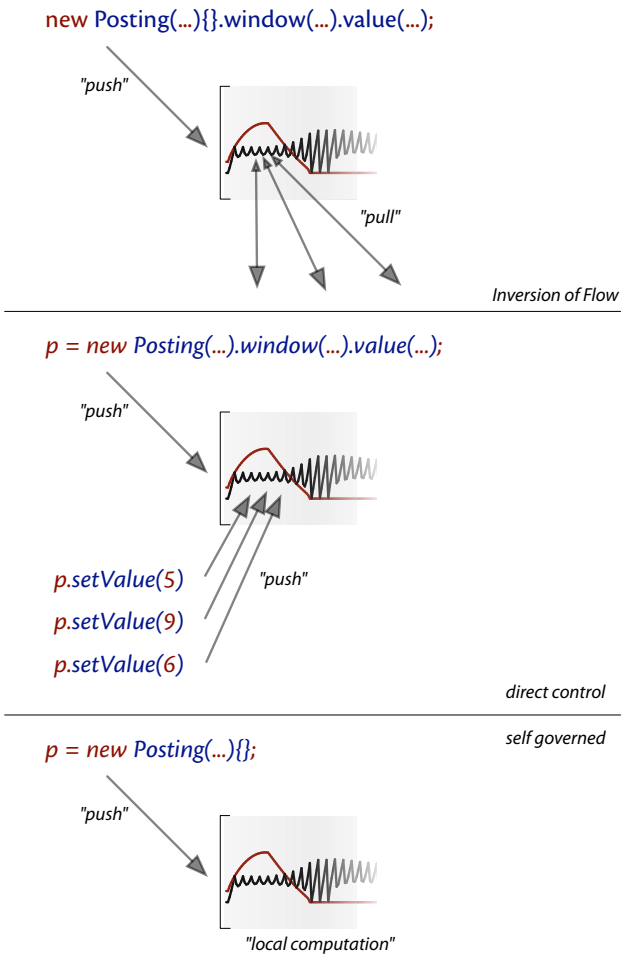


figure 39. Generic radial-basis channels offer three broad categories of flow decoupling.

We can characterize three kinds of relationship:

Ballistic — some postings are sent and then never communicated directly with again. These postings are free to pull the material and data that they need to compute their values and their ending strategy. A posting like this marks an inversion of control flow from an imperative style that pushes data to a location to a single imperative push of *control* to a location that subsequently *pulls* data.

Touchable — Other postings are less independent and the post-*er* frequently sets their value. However, unlike in the conventional forward situation of a purely imperative strategy, the post-*er* is shielded from the burdens of having to set a value on a fixed schedule, or a fixed order. One can construct windowing functions in this case that begin to decay away in the absence of new data — postings constructed with these windows need to be "touched" every so often to keep them valid, otherwise they eventually fade to nothing. This allows a care-free disappearance of the post-*er* and a graceful fade-out of the posting's effects.

Persistent — Finally, some postings are meant never to be culled, and manage the global characteristics of the channel (limiting velocities, smoothing over large changes etc.) or control the behavior of the channel in the absence of any other postings (drifting back to a particular set point, for example).

The generic radial-basis channel forms the foundations of a number of the bodies of the more exotic agents presented in this work (in particular it forms part of the blendable body framework constructed for *how long...* in addition to its service here in *The Music Creatures*).

This broad use is supported by a broad range of value representations. At the level of a point, it's easy to see how a three dimensional position vector can be

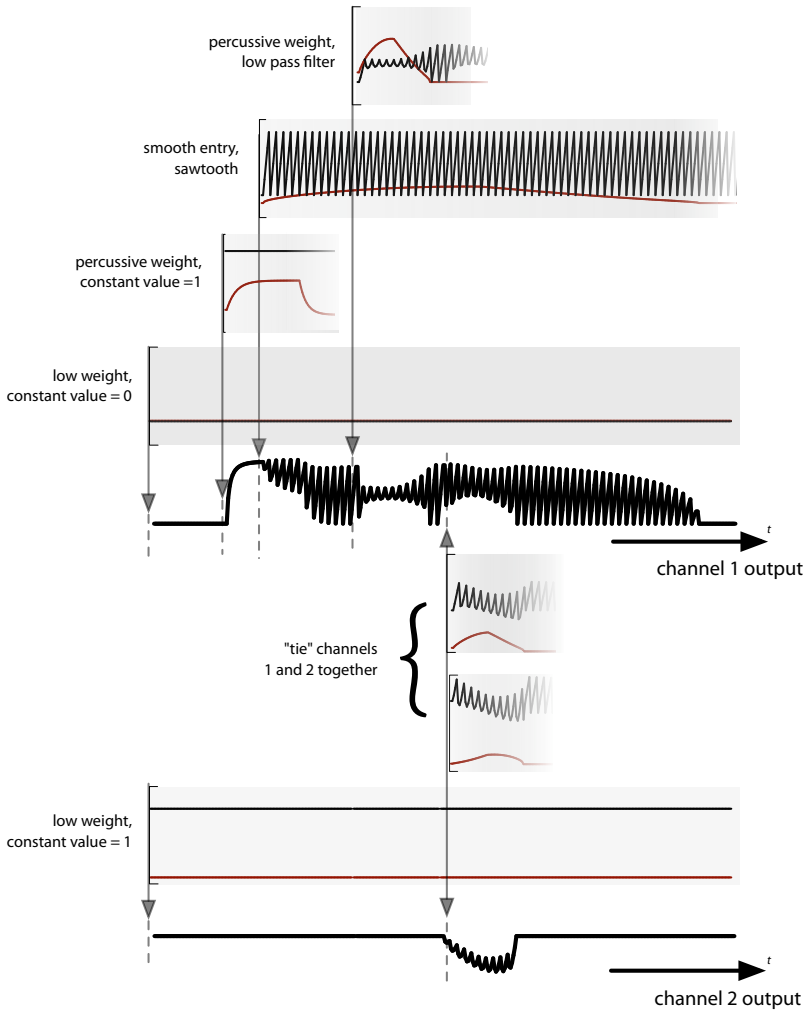


figure 38. The generic radial-basis channel is a place where multiple processes can present small, overlapping, signal processing primitives in an *ad hoc* fashion. In this example, a few postings are added to two channels, including one that tries to temporarily tie the value of both of these channels together.

exposed directly by a value representation interface. More interesting value representations can also be written, with a little more care, for: infinite lines, finite line segments, infinite planes and finite triangles.

For example the piece *22* uses a varying set infinite lines as "fiducial marks". Sometimes these lines couple to geometry in the scene, other times their movement, their momentum, is connected to the movement of the performer. Clearly this is a situation where we have several procedural algorithms operating at different time-scales that need an authorable blending system. The value representation is straight forward:

2d-infinite lines: value representation is a homogenous 5-vector:

$$(x_1\alpha, y_1\alpha, x_2\alpha, y_2\alpha, \alpha) = (x'_1, y'_1, x'_2, y'_2, \alpha)$$

The infinite line goes through (x_1, y_1) and (x_2, y_2). To form $C = Aw + B$ we find the point on A, p closest to the midpoint of B:

$$(B_{x_1} + B_{x_2}, B_{y_1} + B_{y_2})/2;$$

and form two new points, equidistant from p on line A:

$$(A^\star_{x_1}, A^\star_{y_1}, A^\star_{x_2}, A^\star_{y_2})$$

We then blend $A^\star \cdot w$ with B:

$$C = (A^\star_{x'_1} w + B_{x'_1}, A^\star_{y'_1} w + B_{y'_1}, A^\star_{x'_2} w + B_{x'_2}, A^\star_{\alpha} w + B_{\alpha})$$

Using an exponential mapping, we can form a blend space for Quaternions and have orientation valued channels. Scalar representations make channels excellent candidates for modeling the motivations and emotions of an agent — aspects of agents that are good examples of values that are influenced by a variety of systems on a variety of time-scales while also having set points to drift back to. Low-, high- and band pass filters can be constructed that are independent of

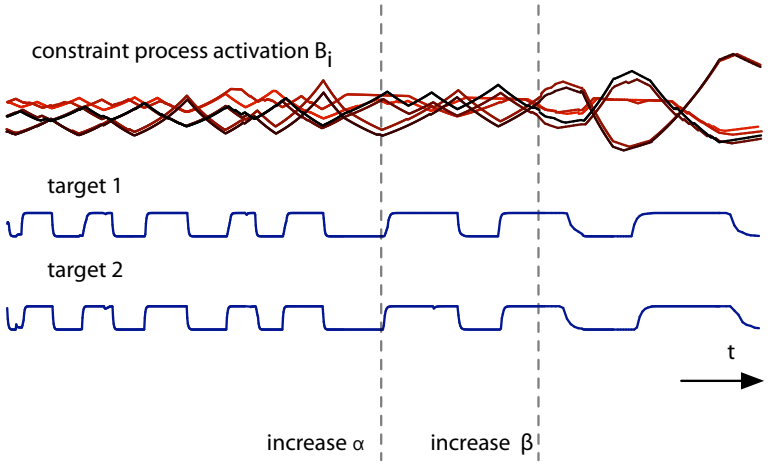


figure 40. In this figure a set of six postings are configured to influence two scalar values. The value representation in this case is a pair of proposed changes to the two targets. Two processes try and make targets 1 and 2 equal to zero; two others try to make them equal to one; and a final pair of postings try to make target 1 equal to target 2 and vice versa. The results of these (over-constrained) processes, as arbitrated by the competitive radial-basis channel structure is to cycle through the two most stable states, targets 1 and 2 having almost the same value for almost all of the time yet oscillating between zero and one.

the value representation, persistent postings can finesse the signal properties of channels, limit velocities. Whole procedural networks can be constructed, tested, and reused in a value agnostic fashion.

Other than window and value generating functions we can extend this basic framework in a number of ways: we can produce exotic value representations and extend the idea of an incremental weighted sum, *page 334*; we can replace the incremental weighted sum altogether with another structure that combines multiple values and weights, *below*; and we can, with a few more constraints on the interface that postings present to the world, allow other processes to inspect, move and organize postings, *page 232*.

The generic radial-basis channel 2 — rediscovering action-selection

Sometimes the degrees of freedom that creatures' bodies offer can be expressed succinctly using these radial-basis channels: for example *tile* doesn't need much more than channels to control its tiles (one channel per tile with a value representation that has parenting information as well as orientation); *exchange* has four rotational degrees of freedom (here, a channel each). But *network* has multiple processes fighting for control over its body. For clarity of movement just one process must be able to gain control of the movement of the network and cause it to move to the lattice. A purely channel based weighted average is a fight without a winner.

Sometimes, then, a weighted average is inappropriate. Rather than blending between two different ideas of what a particular value, position, rotation, line or mesh should be, it's better to pick one and stay with it for a while. Of course, this intuition needs some careful refinement — what constitutes two different ideas of what a particular value should be, and what does it mean to stay with one for a while? No matter how carefully we build the control structure for the

weights, we cannot pick the correct weights for a blended average without looking more closely at the contents of the blend itself.

So we need to extend the value representation to give a little more insight into the material being blended.

```
interface ExtendedValueRepresentation<t_value>{
    t_value zero();
    t_value add(t_value a, float w, t_value b);
➡   float normalizedDistance(t_value value1, t_value value2);
}
```

This additional method provides a normalized distance between two values inside the representation — a distance of 1 means that these values are completely different (that is, two different ideas of a value), a distance of 0 means that they are indistinguishable (that is, they are the same idea).

Now we need a structure that modifies the weights obtained from the postings to the generic radial-basis channel.

Consider formulating a matrix D of normalized distances where each element D_{ij} is the distance between posting i and posting j, with $i, j \in 1..N$ with N as the number of postings. The idea is to use this matrix D to *sharpen* the contrast between the weights of the postings W_i such that incompatible postings are not blended, rather the strongest one wins.

Initially, we set:

$$W' = W$$

We perform the following iteration for $\forall i$,

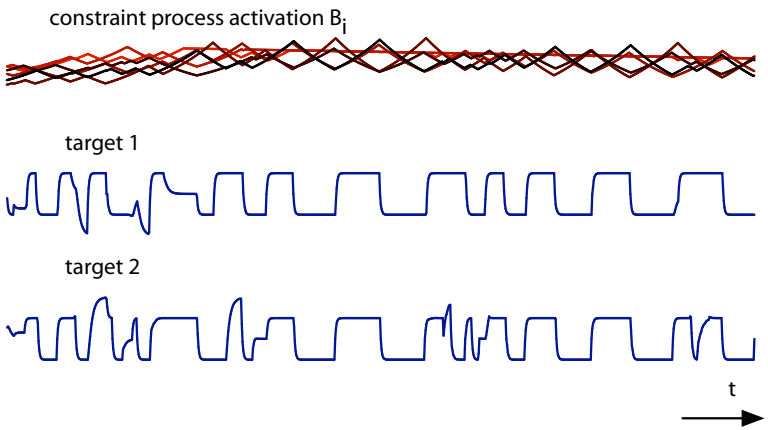


figure 41. A more complex example than the previous case. Six processes again; however two try to make target 1 equal to minus one and one, while two try to make target 2 equal to zero and two; the remaining two are as before, but compete with lower weight. The exploration of the partial solutions to this unsatisfiable constraint problem oscillates randomly, however the two channels remain highly correlated.

$$T_i \leftarrow \sum_{j=1}^{N} W'_j D_{ij}$$

$$W'_i \leftarrow W'_i \cdot (1 - T_i / \max_{j=1}^{N} T_j)$$

$$W' \leftarrow |W'|$$

With each iteration, postings that provide values that conflict with each other, mutually inhibit each other, with their power of this inhibition set by their weights. We can continue this iteration until convergence to produce a winner-take-all weight vector that will either have only a single non-zero entry or multiple non-zero entries that correspond to zero normalized distances. This encapsulates the intuition that we should not blend between "two ideas" of the same value.

Alternatively, we can run a small number, q, of iterations to increase the contrast between weights without removing all of the conflict. Rather than fixing q ahead of time, we might choose a $d_{\max}$ such that $d_{\max} = |W' - W|$ (we terminate should $|W' - W| > d_{\max}$ and set W' such that it is $d_{\max}$ away from W, noting that the furthest away W' could be from W, that is, the largest possible $d_{\max}$, is $\sqrt{N}$)

This may suffice for channels that have many postings that enter and exit in a constant flux. However, for channels with long term postings the temporal dynamics of this algorithm are too clear cut. Stronger postings that conflict always crush weaker postings — no exploration occurs. Two alterations change this:

For the purposes of implementation we must remember this vector notation is strictly mathematical, and in fact N varies with the number of the postings.

firstly, a long term memory, controlled by the time constant α, updated after all of the multiple iterations of the above algorithm:

$$B_i \leftarrow \alpha B_i + (1-\alpha)(1-W_i')$$

with $\alpha \in 0..1$

and, if $W_i' > \varepsilon$ then $G_i = \beta$ else $G_i = 1$, with $\beta \geq 1$

secondly, we use this memory to modulate the weights as they enter the algorithm:

$$W_{i,\text{initial}} \leftarrow W_i B_i G_i$$

Together α and β control the time-constant of changeover that we would expect between two $d = 1$ postings with identical weights. Specifically for large α and large β, changeover will occur slowly, for α close to 1 sensitivity to transient changes of weight and distance is reduced. Specifically, the time constant varies like $\ln(1/(\beta\alpha))$.

Although I have deployed generic radial-basis channel formulation inside motor systems, this extension to the framework, transparent from the point of view of the posting interface and only a small addition to the value representation, actually spans a space of action selection algorithms, where the "action" selected is which (combination of) postings to back in the blended output. Most notably, the space of action selection algorithms spanned by choices of $(d_{\max}, \alpha, \beta)$ includes that of Blumberg ($d_{\max} = \infty$, α and β control the "level of interest" in the language of that model) but also includes models where multiple actions act

B. Blumberg, *Action-selection in hamsterdam: lessons from ethology.* Proceedings of the third international conference on Simulation of adaptive behavior, Brighton UK. 1994.

simultaneously ($d_{\max} \ll \sqrt{N}$, $\beta \approx 1$) and when the transition between actions is soft ($\alpha \approx 1$, $\beta \approx 2$)

Generic radial-basis channels with normalizedDistance(...) value representations are used in the implementation of"soft constraints" in the Fluid representation — multiple processes seek to maintain relationships between the positions and sizes visual elements on a page, these over-constrained set may admit no global solution, yet one would like the to layout both explore the space of possible solutions and to keep close to layouts that are partial solutions. Here the "values" are modifications to individual positions and sizes, and the normalized distance between two displacement vectors V_1 and V_2 is:

$$\max\left(0, -\frac{(V_1 \cdot V_2)}{|V_1||V_2|}\right)$$

Such a metric is also used as the basis for the final output to the body of *network* to ensure that only compatible ways of satisfying its "imagined" physical constraint are attempted simultaneously. The framework allow competitive processes to be added to the generic radial-basis channels. From an authorship perspective, multiple postings can be added to the channel without fear of the system "breaking" under the strain of incompatible request — rather the channel will simply do the best that it can, and while doing so, explore many of its partial options.

This class of channel completes the discussion of the alternative "flow control" strategies developed for *The Music Creatures* and deployed beyond. In the coroutine and resource frameworks we have, at last, a scripting technique for use in defining ordered actions and motor-programs: one that is simple enough to admit a clear, imperative style, but strong enough to survive the uncertainties of interaction and complex system. In the generic radial-basis channels we have a

generic framework for the creation of not only signal manipulation networks, but "process manipulation networks" — for the cases where a symbolic level scripting approach does not "script" at the correct resolution. These channels are for the numeric representations, typically of agents' bodies, for the places where many systems, many lines of code, exert uncoordinated influences over central points. Finally, in the cases where neither a carefully computed blend nor a strictly "action-level" action selection technique is appropriate, I have offered a hybrid — a generic radial-basis channel-based "soft" action selection strategy. All are general purpose techniques for the creation of mid-points, loci of interactions inside complex systems.

5. Concluding remarks — *The Music Creatures'* aesthetics of agency

This chapter began with an overview of the lives of four different kinds of agents and concluded with a detailed description of the techniques constructed to realize these narratives. Despite the ordering of this chapter, the narratives did not completely precede or succeed the frameworks used to create them; rather in constructing the techniques for the principles, the narratives emerged. This interplay between *narrative* and *technique* seems to be a fundamental emergent property of the agent-based installation that involves creatures that possess genuine life-spans populated by periods of learning and development. Specifically, this separates these works from those typically produced under the banner of artificial life. Although oddly shaped and perhaps appearing closer to the machine than the animal, and despite their ultimate disposal and replacement on an almost regular time-scale in the gallery, these creatures are never the anonymous agents that artists' readings of artificial life seem to provoke. Rather they are, even in their limited ways, condensations of some experience particular to their installation space and to the people whose space they share.

Their seven-to-ten minute life-span takes them through a number of changes that, when multiplied by the other creatures, offers a balance of complexity and order that survives in its gallery setting. *The Music Creatures* populate a number of interaction time-scales up to this seven-minute mark: at the smallest level, their performance of sound is coupled to the small perturbations of their graphic lines; they respond promptly to sound made in the gallery; they maintain musical cells that last ten or twenty seconds; they offer visual and behavior changes that take place over a few minutes. On longer time-scales, the whole installation undergoes cycles lasting hours, as material is perpetuated by creatures hearing other creatures. Change over each of these durations seems important for finding an interactivity that remains both direct and lasting.

This said, however well-prepared the balancing strategies involved in the making of this work, they speak little to the problems of constructing a work that unfolds over days, weeks or months. It remains an enticing vista for future work to create agents such as *The Music Creatures* that exist on longer time-scales. We shall revisit this horizon briefly when we consider the future directions for my work.

As a stratagem of indirection, *The Music Creatures* offer an extremely deferred model for the creation of music from the sounds in a gallery. Such *indirection* is a hallmark of the agent-based; the deferral to the agent marks its autonomy and indeed its very agency. In *Loops*, the flux and distribution of finite material and stylistic change is the object of indirection; when we reach *how long...* it is the creation of animation material and meaning-bearing relationships that is indirectly specified in the agent formulations. Yet as we move away from the nearly ambient, or certainly minimalist musical aesthetic of *The Music Creatures*, the complexities and the stakes of this indirection become much higher. As I shall see in *Loops Score* (the next musical work I present), by the time I am ready to approach the problems of music again, I will have had to reinvent the action-

selection architecture of the agent framework to increase the ways in which I can direct this indirection.

Additional aspects of *The Music Creatures* persist in my most recent work for dance, the most striking of which is the music creatures' sense of effort and intention. Each of these creatures has been, in many respects, set up to *fail*. *Exchange's* learning isn't quite strong enough to offer it a stable model of the results of executing a pose; it fails to account for the higher-order physical effects like momentum on the body, and the creature thus over- and under-shoots its target. *Line*'s internal representation of pose is left deliberately over-specified — leaving the creature to struggle from pose to pose. *Tile* is given an impossible task of grasping, indeed embodying, the rhythm of sonic material passed from other creatures that usually have little direct regard for rhythm. *Network* visualizes the frustration of an overly and overtly simple transition model of musical material. Thus, these creatures are in a constant state of imbalance and inadequacy, constructed not of perfectly functioning parts, but rather of pieces that fail and continually re-compensate in interesting ways in unpredictable environments — their own actions add to, rather than negate, their environment's unpredictability. As much as they are constructed with systems that offer long-term learning and the automatic calibration of their perceptual systems, this just serves to keep the creatures on their edge of complexity. That they have technologies with strongly open interfaces, waiting to blend and shift competing signals to their bodies is a symptom that they possess multiple and simultaneously conflicting intentions. Unlike animals such as birds or even characters such as *Dobie*, these creatures are not well-matched to their ecology or installation setting.

It need not have been like this, and it would have been simpler if it had not: I could have built stronger motor learning for *exchange* or given the agent access to the physics simulation so that it could "cheat"; *line* could simply have per-

formed a linear "morph" between poses; *network* need not obey an imagined physical constraint — it could simply twitch in time to its sound; *tile* might have acted to guarantee the rhythmic qualities of the colony.

However, the resulting aesthetics of failure and frustration, of effort and intention, stands in marked contrast to the smooth ease of much of computer graphics and interactive art, and the slick successes of the technology demonstration. That these agents manage to remain, purposefully, at a point of interactive disequilibrium is in itself the technical achievement of the work. I believe this to be one of the destinations of the agent-based, and such traces reappear in my works for dance theater; my agents there chase after all too inadequate and transient perceptions of movement and graphic form.