

# **A framework for comparison of process in algorithmic music systems**

Rene Wooller, Andrew R. Brown, Eduardo Miranda, Rodney Berry, Joachim Diederich

**T**his paper presents a framework being developed to clarify discussion of algorithmic musical processes. We will firstly present some of the current terminological problems then discuss related research and background concepts. Following this, two principal attributes that enable intuitive comparison of algorithmic compositional processes will be defined in detail. The framework will then be demonstrated by applying it to some representative examples of differing algorithmic music systems.

## **1. Introduction**

### *Problems*

The concepts and terms currently applied to discussions of algorithmic music are varied and, especially for new comers, often confusing. For example, consider the term "generative music", which is used with different meanings by various scholars and famous practitioners. The background fields and paradigms of these developers of algorithmic music and their (non-exclusive) interpretations of "generative music" are summarised below:

1. *Linguistic/Structural*: Music created using analytic theoretical constructs that are explicit enough to generate music (Loy and Abbott 1985; Cope 1991); inspired by generative grammars in language and music, where generative instead refers to mathematical recursion (Chomsky 1956; Lerdahl and Jackendoff 1983).
2. *Interactive/Behavioural*: Music resulting from a process with no discernable musical inputs, i.e., not transformational (Rowe 1991; Lippe 1997: 34; Winkler 1998).
3. *Creative/Procedural*: Music resulting from processes set in motion by the composer, such as "In C" by Terry Riley and "Its gonna rain" by Steve Reich (Eno 1996).
4. *Biological/Emergent*: "Non-repeatable music" (Biles 2002a) or non-deterministic music, such as or wind chimes (Dorin 2001), as a sub-set of "Generative Art".

Clearly, there are various ways to discuss the same phenomena. It is the goal of this paper to initialise development of a more integrated and

hopefully less confusing framework, especially useful for comparisons of algorithmic music systems in their own right. We emphasise comparison because, while discussions of process in any single system will naturally occur within a specialised context, dealing with multiple systems will require a way of relating them through a shared perspective. With this in mind, we will try to draw together some of the diverse concepts into what, we hope, is a simple, flexible and extensible framework for discussing musical algorithms.

### *Background*

A clear conceptual framework for developers of algorithmic music does not exist; however there are many publications that discuss algorithmic music to greater or lesser extent and all of them assume some kind of conceptual basis. In this section we will review some of the literature to clarify various existing schools of thought. However, our notion of "algorithmic music" should be defined first of all. Simple algorithmic or procedural music has existed from well into the middle ages (Roads 1996), however, this work is only of historical significance to the field. We are instead concerned with the kind of computational research that is generally intractable without a "high-speed digital computer", as originally carried out by Hiller and Isaacson (1958).

Barry Truax (1976) proposed a conceptual framework for music theorists based on careful observations of musical practices. The predisposition of realtime computer music to this form of research was made clear. Truax felt the procedural focus was primarily about relationships between system components (including the human) and could be termed "communicational". We support the notion of viewing algorithmic composition as utilizing a system of related components, but choose to borrow terms from programming rather than sociology and psychology; for example, see "Levels of encapsulation" below.

The "Generative Theory of Tonal Music" (Lerdahl and Jackendoff 1983) examined the psychological basis of hierarchical structures conceived in homophonic tonal music and has significantly influenced the formalization of music generally. The authors are careful to present their generative theory as an aid to music analysis, rather than an "algorithm that composes a piece of music" (p.5). The connection to the work of Chomsky is in the "combination of psychological concerns and the formal nature of the theory" (p.5), rather than any particular aspect.

Before any further reference to linguistics, we will briefly explain relevant terminology, initially presented by Chomsky (1956), where the focus was on structural forms. For Chomsky, a transformation "rearranges the elements ... to which it applies, and it requires considerable information on the constituent structure" (p.121). Generative was conceived as the property of a finite set of rules that

could potentially create, via recursion, a huge or infinite output with a characteristic structure. For example, he examined different forms of rules, seeking to find one that could “provide simple and revealing grammars that generate all of the sentences of English and only these” (p.113). In music compositional terms, generative grammars capture the essential logic of particular “top-down” approaches.

In reviewing a number of sound and music programming languages, Loy and Abbott applied programming language and linguistic concepts (1985). The potential of Object Oriented (OO) concepts applied to computer music was identified (p.262-263). Algorithmic music endeavours were distinguished by the level at which the composer influences the code (1985: 238), for example tinkering with programs, writing libraries or creating languages. This approach highlighted the amount of compositional control but not the different relationships between musicality and procedural forms (which was not the intent of the authors).

Roads (1985) compiled articles to represent the views of composers working directly with computers. The range of techniques and aesthetic considerations are organised into various topics. Of these “procedural composition” is the most relevant to us, where the composer-programmer is less concerned with audible outcomes as they are with the algorithms behind them. However, to Roads, the musically innovative aspects of composing with computers dealt with sound structure rather than traditional concepts such as note organisation (p.xviii). This view is contrasted with ours in “Perspective”, below.

Pope (1991) reviewed systems that utilise OO technology, thus focusing on various programming tools and their idiosyncrasies rather than the contrasting musicological trends of algorithm designs and mappings.

Desain and Honing (1992) adopted a perspective combining music theory, music psychology and Artificial Intelligence (AI) primarily based on music cognition. The insight that “music is based on, plays on, and makes use of the architecture of our perceptual system” (p.6) led this and subsequent studies (Desain et al. 2005) into the development of *generalised* musical representations and algorithmic musical processes, especially concerned with analysis. Without being prescriptive, Desain and Honing are careful to limit the scope of their search, necessarily ignoring many aspect of music. However, their view facilitates deep examination of musical processes by liberating them from idiosyncrasies and enabling comparisons in a general conceptual space, grounded in the architecture of the mind.

Schwanauer and Levitt (p.1-6) conceive algorithmic musical processes as machine models of music and as useful tools for scientific examination of theories. Unlike Desain and Honing, emphasis is placed

on AI as an increasing source of inspiration for procedural music, with little acknowledgement given to the influence of music theory, practice and psychology. AI, as a computer science perspective, facilitates discussion of efficiency but neglects the musical purposes of algorithms.

Roads explains algorithmic composition systems and representations in the Computer Music Tutorial (1996: 821-909). Systems are discussed in terms of *history*, *presentation* to the user, *interactivity* and level of *composer responsibility*. A special distinction is made between *deterministic* and *stochastic* algorithms. Roads also states that, apart from simple cases, there is no perceivable difference between the two (p.836). Dodge and Jerse (1997: 341) also examine some "simple cases" and draw the same distinction. We debate Roads' statement and the implications in "Limitations", below.

Laurson introduced the laudable but unreachable aim of "musical-neutrality" for a music software environment in his discussion of PatchWork (1996: 18), as a state where there is no difference between compositional goals and outcomes. We broaden the term to also mean an absence of musical side effects in any implementation of a musical process.

Central to the model of interactive music systems presented by Winkler (1998: 6), are domains of algorithmic music, namely: music analysis, interpretation, composition. Winkler presents his ideas relating to algorithmic composition through Max (Puckette and Zicarelli 1990), a very specific framework for discussing musical process.

In reviewing AI techniques applied to music, Papadopoulos and Wiggins (1999: 1) admit to difficulty in distinguishing works according to the techniques used within them. As this was the problem that led us in this paper to investigate more appropriate ways of conceiving algorithmic music, it will now be reiterated:

The categorisation is not straightforward since many of the AI methods can be considered as equivalent: for example *Markov chains* are similar to type-3 grammars. Furthermore some of the systems have more than one prominent feature, for example *EMI* (see below) was categorised as a grammar, but it can also be seen as a knowledge-based system or even a system which learns.

In our proposal we deal with comparative problems through relativistic descriptions of process, as outlined in "Introducing the framework". Also, it seems necessary when comparing algorithmic music systems to relate the processes back to musical experience, as discussed in "Representing music and processes".

## *Perspective*

As shown in the previous two sections, algorithmic music is discussed from a variety of perspectives, often combined: linguistic, musicological, sociological, user-centric, artist and programmer-centric (to name a few). This framework is tailored exclusively for the last two, that is, programmers/composers of algorithmic music. These people alone are necessarily occupied by executably explicit formalizations of musical processes. Although many algorithmic composers consider intuitive interference to be of equal or greater importance to formalization, it is clear that at some stage process *must* be dealt with. A specialised frame of reference with clear terminology would be useful when this stage is reached.

It is important to note that the perspective of the algorithmic music developer is synergistic and requires a unique although obviously multi-disciplinary outlook. We have hopefully shown in the section on background that when emphasis is placed heavily on particular related fields, unexpected complications in explaining and comparing algorithmic music systems can arise.

## *Limitations*

This framework is designed to help developers discuss similarities and differences between algorithms and how they relate to musical results. However, there are some universal limitations that must first be acknowledged.

As has been said before, no implementation of an abstract process can be musically neutral. However, we should now consider that *no abstract process can be definably musical*, a point touched on when discussing Roads, above.

Despite this, some musical processes are easier to implement than others, for example one-to-one mappings are relatively trivial, which explains the observation that certain processes afford certain musical outcomes.

As well as this, when a system is interactive normally hidden processes become much more transparent to the audience/user; especially if one is conscious of the input into the algorithm or the seed parameters. Thus, for adaptive and interactive systems, addressing the effect of process on music is no longer merely a matter of "compositional philosophy and taste" as Roads claims (1996: 836), but an aesthetic and usability imperative. This framework may therefore be particularly useful for designers of interactive systems.

## 2. Representing music and processes

If, as stated, algorithmic processes influence the musical experience then, similarly, musical representations influence the use of algorithmic processes and their ease of implementation. Stated more formally, the knowledge representation denotes the ontology of the algorithmic music system designer and carries with it the insight and blindness of that viewpoint (Balaban 1997; Brown 1999). Although processes are the primary subject of this paper, the way we address representational implications such as mappings, encapsulation and musical predisposition must first be explained.

### *Mappings*

The greater the similarity of musical mappings across algorithmic music systems, the easier they can be compared. Thus, for our framework to be applied effectively, it is first of all necessary to identify and differentiate this aspect. If they are similar, one can then progress to a comparative discussion of musical process. If they are significantly different, a framework for representation schemes (Wiggins et al. 1993), may be more appropriate. Alternatively, specific modules of the system that deal with comparable music representations could be identified. At least, any differences between systems that might be attributable to variations in the mappings should be taken into account.

### *Levels of Encapsulation*

The way we encapsulate processes greatly affects the way they are conceived, and thus discussed and compared. The term *encapsulation* is drawn from the language of OO programming. It is a method of viewing a potentially complex process as a simple component (capsule) with inputs and outputs, enabling the programmer to ignore the complexity of the system as a whole and focus on a specific portion.

When using this framework, attention must be paid to the level of encapsulation being applied to the algorithmic music systems analysed. We do not prescribe any particular encapsulation scheme, but provide two scenarios and present the different outcomes, below. Briefly, encapsulating the system as a whole tends to filter the detail, attenuating the range of classes encompassed by the system. Conversely, encapsulating sub-systems and adding their individual descriptions together leads to more detail and broader descriptions. We do not prescribe any particular encapsulation scheme, but provide examples that use the latter.

### *Musical predisposition of representations*

At a conceptual level, musical predisposition is a measure of how well the musical representation can portray the musical ideas and facilitate the compositional processes that are required. If one were so inclined, this might be derived formally by ascertaining the lack of complexity in applying the representation to specific compositional processes. Gauging algorithmic complexity itself is a well defined area of computer science (Russell and Norvig 2004) and is outside the scope of this paper.

A representation may become more predisposed to a certain musical process by adding relevant parameters or structuring the representation more appropriately. For example, in a note representation that only consists of duration and pitch, adding a vibrato parameter would increase the predisposition towards musical processes that incorporate expression. Representational structure is important also; consider the difference between a sequence (ordered) and a set (unordered) of note events. Without some computation, it is impossible for the set to represent the musical structure of a sequence containing the same events. Therefore, we maintain, sets of individual notes have less predisposition to musical tasks such as "create an eight bar phrase" than do sequences which may already be part way organised as a phrase. On the macroscopic level, a large database of licks (set) is not yet a piece of music but a sequence of licks can be.

Many other sonic, performative and structural elements affect the musical predisposition of representations and have implications for the nature and potential of the algorithms using them however, space does not allow for a full discussion of them at this time.

We acknowledge that musical predisposition, if implemented as an absolute measure, fails to represent the specialised stylistic aspects of disparate musical implementations. Instead, our framework only ever uses predisposition as a relational indicator. That is, a representation can only have "more or less" general musical predisposition, as is made clear when analysing (not prescribing) the function of processes in the main body of the framework, below. Thus, a systematic method for obtaining the general musical predisposition of a representation in relation to a specific group of algorithmic music systems could be (in future research) to define its musical predisposition to each of the algorithms and take the average.

### **3. The framework**

Having established the elements that represent musical processes and data and the associated notion of musical predisposition, we can now present a framework for positioning algorithmic music systems along two main dimensions, function and context.

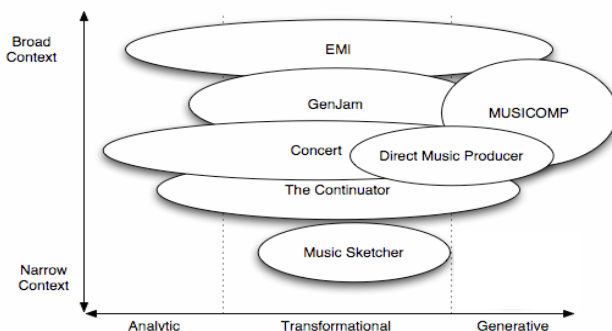


Figure 1. Some programs located within the algorithmic music framework.

### Function

In our framework, we position musical algorithms on a continuum describing function that ranges from *analytic*, through *transformational*, to *generative* (the x axis in Figure 1). These terms describe the effect of the process upon the data applied to it. There are two ways process can be seen to modify data:

1. Change in potential size of data.
2. Change in representation scheme.

Both of these effects are used by us to indicate the location of the process on the functional continuum. Potential size is considered to be the collective size of all possible outputs. If the output is potentially infinite, some creativity may be needed. For example, one could measure the amount of information in the representation to give a further indication of size. Alternatively, the most typical sizes that occur in practice could be used.

**Analytic** algorithms tend to reduce the potential data size and the general musical predisposition of the representation by extracting specific features. For example, a process that takes a set of sequences and outputs a set of notes can be considered analytic. The set of notes could be a scale that has been distilled from a database of riffs. Another example could be the Shenkerian process of deriving a pitch progression, or *urlinie* where a polyphonic piece of music is distilled into a single monophonic line (Lerdahl and Jackendoff 1983; Cope 1996).

**Transformational** algorithms tend not have a significant impact on the general musical predisposition of the data representations or the actual size of the data, but can alter the information. For example, an algorithm that transposes individual notes retains the parameters and structural relations of note collection as representation, but alters the pitch value of each note. A retrograde algorithm that reverses the order

of a phrase retains the sequential note representation while transforming the structural pattern; but, in the end, the general musical predisposition of the transformed phrase is unaltered.

**Generative** in the context of this framework means “musically generative” and is defined as follows. An algorithm tends toward being generative when the resulting data representation has more general musical predisposition than the input and the actual size of the data is increased. For example, a chaos music algorithm that takes a number as a seed and returns a sequence of notes is generative.

The method of encapsulation influences the size and position of a musical system on the continuum. Encapsulating sub-system components and representing their different functions as a collective may lead to a broad definition. Encapsulating the whole system means that it occupies a single point. Other considerations when determining functional location include what counts as input. For example, input might occur interactively during runtime, but does it also occur when the musical material is prepared before-hand? In the software, where is the line between data and process? We apply the sub-systemic approach; however it is up to the user of the framework to decide which boundaries will lead to richer comparisons for aspects of particular interest.

### *Contextual Breadth*

Context is the surrounding information that influences the computation of an algorithm and therefore an algorithmic music system. In OO programming terms, the context is the world-state data and arguments that the algorithm has access to. Context has a breadth or size which can extend along a continuum from localised processes that are context-free to processes that are highly context dependent. This definition of context dependency encompasses Chomsky’s (Chomsky 1957), but is applied more generally to non-symbolic techniques.

An intuitive way of visualizing contextual breadth is to imagine a note positioned on a musical score with a circle drawn around the note that defines the scope of its context. The circle would include previous and future notes in the same part and concurrent notes in adjacent parts. The size of the circle equates to the breadth of context. In practice the contextual breadth has more than these two (temporal and textural) dimensions. For example, an algorithm where notes are influenced by four parameters has a broader context than a similar algorithm that considers only one parameter.

Optimally, all processes used in an algorithmic music system should be examined when discussing them, however, most applications are too complicated to fully examine within the limited space of this paper. So,

in the remainder of the paper, we will demonstrate how the framework assists in general discussion of algorithmic music systems, by describing the salient features of a number of applications and how they might be positioned within the function-context framework. Refer to figure 1 to see where we have positioned each system within the framework.

## **4. Example applications**

### *MUSICOMP*

MUSICOMP is the software outcome of the earliest comprehensive investigation into algorithmic composition using a computer, started by Lejaren Hiller and Leonard Isaacson in 1955. Four sets of experiments conducted throughout the project, each exploring different aspects of the problem of applying the computer to music (Hiller and Isaacson 1958).

The general procedure in the first two experiments was to modify and constrain "random white-note music" with rules, mostly inspired from counterpoint harmony. The random process has a narrow context and is generative. Pre-composed material is sometimes involved, for example the insertion of a C chord at the beginning and end, or cadence completion. This is viewed as a transformational process at the level of musical structure, with some breadth of context required to determine exactly where to put it. The other rules such as "No parallel unisons, octaves, fifths" or "no more than one successive repeat" are transformational but with a narrower context, in that no additional information is needed beyond the notes at a single beat that are being operated on. The third experiment applied the same generate-modify-select structure to a contemporary musical aesthetic.

The fourth experiment used Markov processes and in some cases a system for relative tonality using a closed musical structure. Features such as consonance and dissonance could be controlled by affecting the influence of various probability distributions. The tonality system provided an additional increase in contextual breadth by marking notes for tonal reference and relating the following notes to it. Because the Markov table was constructed manually, we did not consider the system to be analytical even though one that did automatic Markov matrix construction would be.

### *Experiments in Musical Intelligence (EMI)*

EMI has been developed by David Cope since 1981, with the aim of replicating musical style. At a conceptual level, the work is an exploration of creative musical process, focusing in particular on recombination and grammars (Cope 1996; Cope and Hofstadter 2001). The former is interpreted as a transformational process that occurs at

the level of musical phrase and it is clear that the latter has generative properties.

In general, global musical structure is manually incorporated into EMI, while phrase and measure level organisation is generated by Augmented Transition Networks (ATNs), and note-level organisation by Micro-Augmented Transition Networks (MATNs). These grammars provide rules governing which segments are allowed to follow other segments. A manually encoded framework determines general principals of rules, while the specific combinatorial constraints are determined through analysis of the database of music.

The complex systems of grammar used in EMI, such as SPEAC, add many dimensions of input to the algorithms, denoting a broad context.

### *CONCERT*

Mozer experimented into note-by-note prediction using the CONCERT system (Mozer 1994), aiming see if a connectionist system, without explicit representations of form, could learn structural trends and utilise them in coherent compositions. Listener testing found that, while the compositions were preferred over that of a third-order Markov chain, global coherence was lacking. Mozer concluded by doubting the potential of note-by-note prediction, but offered suggestions for future research.

To compose music, the system is initially trained to predict the next note in a sequence or training set. The first notes of a sequence are then given, and predictions are fed back recursively as input, thus predicting an entire composition. The pitch representation was interesting in that it attempted to reflect psychoacoustic similarities between parameters. For example, pitch was collectively represented through pitch height, a circle of chroma, and a circle of fifths.

CONCERT has a broad context – in fact a “context layer” of artificial neurons are devoted to this. However, the data within the context layer becomes diluted over time. In the analytical training mode, CONCERT takes a musical sequence and establishes node weightings. However, in the more generative compositional mode, the system is given a starting sequence of limited length and creates a much longer sequence.

### *Music Sketcher*

Music Sketcher is a technology preview from IBM (IBM-Computer-Music-Center 1999), now abandoned. It can be conceived of as a meta-sequencer in that the user applies pre-composed blocks of musical data called riffs. Transformations to musical parameters such as velocity, duration and pitch are applied to riffs via graphs controlling primitive arithmetic operations. The harmony track has the chord progression and

tonality. The "Smart Harmony" system (Abrams et al. 2000) performs an analysis on the original riff, and re-maps the pitches to suit the tonality and chord.

Music Sketcher is based on transformational procedures: musical patterns can be rearranged by a number of functions which do not affect the musical predisposition of the data representation or potential data size. "Smart Harmony" involves analysis to transform absolute pitch to harmonic pitch and generative processes to render it back. The set of riff transformations have a narrow context, affected only by the musical data being operated on and the value of the relevant parameter. "Smart Harmony" has a mildly broad context; despite dealing with both the tonality, harmony and all of the pitch data within the tracks, other aspects such as rhythm and structure are not involved.

### *DirectMusic Producer*

DirectMusic Producer (DMP) (Microsoft 2001) is a tool for composing non-linear music, especially for interactive contexts such as computer games (Buttram 2004). Infrastructure exists for sequencing, tonality, stochastic harmonic progressions, control over musical structure and many others.

The DMP "segment" defines a section of music that renders differently on each playback depending on composer defined settings. The segment can hold different kinds of "tracks". A "Sequence Track" defines a linear sequence and "Pattern Track" defines a track with a bundle of user defined "variations", which are flipped between by the playback engine. Functional chords (I, vi, etc.) that the variation is limited to can be specified. "Motifs" can be defined for occasionally triggered sequences.

The DMP system of grammar allows for a fairly high level of relatedness between data and the algorithms that use it, denoting a broad context. There is no analysis, and the only generative algorithms are random chord traversal and variation selection. DMP relies primarily on recombination of patterns, which is a form of transformation at the level of structure.

### *The Continuator*

Recently developed by Pachet, the Continuator is a virtual improviser that continues musical phrases (Pachet 2002a). Preliminary tests found that the human player is indistinguishable from the virtual in most cases (Pachet 2002b). This may be due to higher level musical structure being less important when the system is only required to produce short continuations. In this interactive context, unlike Mozer (above), note-by-note prediction proves suitable.

The continuator learns a Markov model of possible note sequences from a database of music, which can be built on the fly. A key feature is the use of a realtime 'fitness function' to influence probabilities. The weightings of nodes can be adjusted in favour of notes with similar properties to the input stream provided by the interacting musician.

Pachet has approached the contextual problems related to the application of sequential models to polyphonic music by clustering notes within the same temporal region. Some of the clustering modes are explicit, for example 'fixed metrical structure' clusters notes into beat long segments, regardless of their rhythmic structure, while other modes use a parameter to determine the cut-off point between overlapping notes and chords.

The Continuator uses analytic, transformational and generative techniques – analysis of musical sequences creates sets of note sequences and probabilities, rhythmic re-mappings transform the music and continuations are generated from probability sets. Because the Continuator is influenced by a musical history, musical input and a large number of instructional parameters, the context is fairly broad. However, certain modes are much narrower than others making it difficult to classify context position in an accurate way.

### *GenJam*

GenJam is a virtual jazz improviser, with the capacity to perform solos, trade fours and provide harmony to a human player (Biles 2002b). A repository containing structural and motivic data for the entire standard jazz repertoire provides the algorithm with a predetermined global structure around which to operate.

Local musical structure is derived from recombination and selection. Depending on the mode of operation, material for recombination is either derived from analysis of a general lick database, a database of licks specific to the current song, or human input. The pitch intervals are mapped to a chord progression and scale. The original version was an Interactive Genetic Algorithm (IGA), where the user determined the fitness of the lick combinations. Depending on the mode and level of operation, the newer version of GenJam can randomly combine phrases or uses a function to ascertain the most rhythmically appropriate crossover point. A number of other heuristics have been used to ensure a certain musicality. Biles claims that this avoids the fitness function, although this seems a semantic point; *some* analytical process ensures certain children are preferred. With finite licks and potentially infinite musical output, the system is generative also (Biles 2002a).

Bile's use of these techniques shows how well-chosen transformational processes can impart a sense of musicality as well as providing musical

flexibility. The system also outlines how data-driven recombination techniques can be used to great effect when the stylistic boundaries are narrowed to a single genre.

## 5. Conclusion

Algorithmic music systems can be usefully examined through the lens of the function-context framework presented in this paper. This framework has two dimensions and applications can be mapped onto this space as a way of describing their features and character. We have argued that the potential of musical processes is strongly influenced by the way musical data is represented to an algorithm and that the potential for algorithmic processing and compositional use can be summarised as the musical predisposition of data.

Three types of algorithmic functions have been distinguished within the framework: analytic, transformational and generative. These are distinguished by the way they effect the musical predisposition and size potential of the data that passes through them. The scope of influence on the algorithmic process is the other important dimension of our framework, which we refer to as the algorithm's breadth of context.

Our framework for the comparison of processes in algorithmic music has several applications. It can help composers understand the processes and potential of algorithmic systems when designing or selecting algorithms that meet creative needs. It can assist musicians and researchers to differentiate between algorithmic processes and more clearly articulate the features of computer music systems. Unexplored applications of the framework could include tracking the evolution of algorithmic music systems and spot design trends. The framework, through extension, might also be applied to system evaluations. We look forward to seeing in what other contexts uses are found for applying this comparative framework.

### References

Abrams, S., R. Fuhrer, D. Oppenheim, D. Pazel and J. Wright. 2000. A Framework for Representing and Manipulating Tonal Music. In *Proceedings of ICMC 2000 International Computer Music Conference*, San Francisco: ICMA.

Balaban, M. 1997. The Musical Structures Approach to Knowledge Representation for Music Processing. *Computer Music Journal*, 20 (2): 96-111.

Biles, A. 2002a. GenJam in Transition: from Genetic Jammer to Generative Jammer. In *International Conference on Generative Art*, Milan, Italy.

Biles, A. 2002b. GenJam: Evolutionary Computation Gets a Gig. In *Third Conference on Information Technology Curriculum*, Rochester.

- Brown, A. R. 1999. Tools and Outcomes: computer music systems and musical directions. In *The Imaginary Space: The Australasian Computer Music Conference*, 16-22. Wellington, New Zealand.
- Buttram, T. 2004. Beyond Games: Bringing DirectMusic into the Living Room. In *DirectMusic?* ed. Fay, T. M. Plano, Texas, USA: Wordware Publishing, Inc.
- Chomsky, N. 1956. Three models for the description of language. *IRE Transactions on Information Theory*, 2: 113-124.
- Chomsky, N. 1957. *Syntactic structures*. The Hague, the Netherlands: Mouton.
- Cope, D. 1991. *Computers and musical style*. Madison, Wis.: A-R Editions.
- Cope, D. 1996. *Experiments in Musical Intelligence*. Madison, Wisconsin: A-R Editions.
- Cope, D. and D. R. Hofstadter. 2001. *Virtual music : computer synthesis of musical style*. Cambridge, Mass.: MIT Press.
- Desain, P. and H. Honing. 1992. *Music, mind and machine : studies in computer music, music cognition and artificial intelligence*. Amsterdam: Thesis Publishers.
- Desain, P., H. Honing, J. A. Michon, M. Sadakata, Y. Schouten and P. Trilsbeek. 2005. *Music Mind Machine Publications*. <http://www.nici.kun.nl/mmm/publications.html> (accessed 14th of February, 2005).
- Dodge, C. and T. A. Jerse. 1997. *Computer Music*. 2nd ed. New York: Schirmer Books.
- Dorin, A. 2001. Generative processes and the electronic arts. *Organised Sound*, 6 (1): 47-53.
- Eno, B. 1996. *Generative Music*. <http://www.inmotionmagazine.com/eno1.html> (accessed 27th of July, 2005).
- Hiller, L. and L. Isaacson. 1958. Musical Composition with a High-Speed Digital Computer. In *Machine Models of Music*.
- IBM-Computer-Music-Center. 1999. *Music Sketcher*. IBM. (accessed 12th of December, 2004).
- Laurson, M. 1996. *Patchwork*. Helsinki: Sibelius Academy.
- Lerdahl, F. and R. Jackendoff. 1983. *A generative theory of tonal music*. Cambridge, Mass: MIT Press.
- Lippe, C. 1997. Music for piano and computer: A description. *Information Processing Society of Japa SIG Notes*, 97 (122): 33-38.
- Loy, G. and C. Abbott. 1985. Programming languages for computer music synthesis, performance and composition. *ACM Computing Surveys (CSUR)*, 17 (2): 235-265.

- Microsoft. 2001. *DirectMusic Producer 5.3.0.900*. Microsoft.  
<http://www.musicmachines.net/> (accessed 8/03/04).
- Mozer, M. 1994. Neural network music composition by prediction: Exploring the benefits of psychoacoustic constraints and multiscale processing. *Connection Science*.
- Pachet, F. 2002a. The Continuator: Musical Interaction with Style. In *ICMC*, 211 - 218. Göteborg, Sweden.
- Pachet, F. 2002b. Playing with Virtual Musicians: the Continuator in Practice. *IEEE Multimedia*, 9 (3): 77-82.
- Papadopoulos, G. and G. Wiggins. 1999. AI Methods for Algorithmic Composition: A Survey, A Critical View, and Future Prospects. In *AISB'99 Symposium on Musical Creativity*, Edinburgh.
- Pope, S. T. ed. 1991. *The well tempered object*. Cambridge, Mass.: The MIT Press.
- Puckette, M. and D. Zicarelli. 1990. MAX. Opcode. <http://www.cycling74.com> (accessed).
- Roads, C. ed. 1985. *Composers and the computer*. Los Altos, Calif.: William Kaufmann.
- Roads, C. 1996. *The Computer Music Tutorial*. Cambridge, Massachusetts: MIT Press.
- Rowe, R. 1991. Machine Learning and Composing: Making Sense of Music with Cooperating Real-Time Agents. Thesis from *Media Lab*. Mass.: MIT.
- Russell, S. and P. Norvig. 2004. *Artificial Intelligence: A Modern Approach*. 2 ed. New Jersey: Prentice Hall.
- Schwanauer, S. M. and D. A. Levitt eds. 1993. *Machine Models of Music*. Cambridge, Mass.: The MIT Press.
- Truax, B. 1976. A Communicational Approach to Computer Sound Programs. *Journal of Music Theory*, 20 (2): 227-300.
- Wiggins, G., E. R. Miranda, A. Smaill and M. Harris. 1993. A framework for the evaluation of music representation systems. *Computer Music Journal*, 17 (3): 31-42.
- Winkler, T. 1998. *Composing Interactive Music*. Cambridge, Massachusetts: MIT Press.