Contemporary Music Review
Vol. 28, No. 2, April 2009, pp. 133–147

Routledge
Taylor & Francis Group

Interfacing Manual and Machine Composition

Torsten Anders and Eduardo Reck Miranda

Computer-aided composition (CAC) is situated somewhere in the middle between manual composition and automated composition that is performed autonomously by a computer program. Computers cannot make aesthetic decisions in their own right. They can only follow orders. Aesthetic decisions are made by composers, both via the design of computer programs and by manually controlling these programs. The latter plays an important part in CAC. The composition process typically involves much emending and revising: changing how a computer program is controlled is easier and allows for a more intuitive way of working than changing the program itself. This paper argues that constraint programming is a particularly suitable programming paradigm for flexibly interfacing manual and machine composition.

Keywords: Algorithmic Composition; Computer-aided Composition; Constraint Programming; Microtonal Music; Strasheela

Computer-aided Composition

Computers have been used for a long time for music composition; the computer-generated string quartet *Illiac Suite* (Hiller & Isaacson, 1993) shows that this field is almost as old as computer science. Systematic surveys of the field are provided by Roads (1996), Miranda (2001), and Taube (2004). Assayag (1998) outlines the history of computer-aided composition, while Berg (2009) surveys some classical algorithmic composition systems.

Computers have been used with different approaches and goals for composing music. Some practitioners of this field lean more towards *automatic composition*, where in an extreme case a composition is generated by pushing a single button. By contrast, others are more interested in *computer-aided composition* (CAC), where composers manually shape certain aspects of the resulting music. The following paragraphs compare these two approaches.

Often, the primary focus of automatic composition is on some composition algorithm or system, rather than on the actual music it produces. An autonomous

ISSN 0749-4467 (print)/ISSN 1477-2256 (online)
DOI: 10.1080/07494460903322422

composition process is used as an experiment for evaluating the system—manually fine-tuning the result could be seen as tampering with the evaluation. Nevertheless, even researchers commonly improve the musical quality of their output by cherry-picking results. For example, CHORAL—a system for generating choral harmonisations in the style of Johann Sebastian Bach—received much attention for the musical quality of its output. Its author, Ebcioglu (1987), concedes: 'The following harmonisations have been manually selected, but from only a few versions for each chorale'. Cope (1996) also happily admits he cherry-picks results of his highly regarded composition system EMI '[...] the music I finally choose to release has been selected from a large number of non-acceptable works. I ultimately choose music that I personally find the most convincing' (Muscutt, 2007).

Automatic composition is interesting for applications that make it impossible for composers to intervene while music is generated. For example, Essl's (2000) *Lexikon-Sonate* was originally conceived as a musical commentary to a hypertext novel *Lexikon-Roman*, where readers freely navigate through the text and the music follows automatically. Video games are another area with great potential for automatic composition, although its application is still in an early stage of development in this field (Collins, 2009). Also, some composers are genuinely interested in a highly automated approach to algorithmic composition (Collins, 2008).

Most composers, however, are keen on shaping the generation process of composition software according to their aesthetic preferences. As a first measure, composers manually edit algorithmically generated music. Koenig (1980) argues that composers should manually 'interpret' the results returned by a composition system, instead of leaving the raw results unedited. Nevertheless, it should be mentioned that Koenig's compositional practise of interpreting results was also due to technical limitations of his software, which generated tables instead of music notation (Essl, 1989).

When doing computer-aided composition, composers apply or even develop certain technical means, but these technical means are not the actual purpose; the main result of their work is the artistic output. Composers decide which compositional parts or aspects of the music the computer generates, and which parts are composed manually. For example, composers may write some rhythmic phrases by hand, then generate pitches and articulations for these rhythmic phrases with a computer program, and finally arrange the phrases by hand into a composition. Using such an approach the computer becomes a 'silicon assistant' for the composer. In the seventeenth century, eminent painters often ran workshops with apprentices and students who helped the master (Fleischer & Scott, 1997). The master only cared for the important contents of the paintings—typically hands and faces—while workshop members carried out the rest. With a grain of salt, in CAC today the computer can take over the role of such a workshop that assists the composer. Perhaps an even better comparison is that in CAC, the computer can act as a 'bicycle for the mind'. Although only the person who rides it powers a bicycle, it greatly improves the mobility of that person. Likewise, using computers can lead to

music that would have been much harder or even impossible to compose manually. Nevertheless, computers cannot make aesthetic decisions on their own right. They can only follow orders. Aesthetic decisions are made by composers, both through the design of computer programs and by controlling these programs with arguments.

Laske (1981) characterises CAC as follows: 'We may view composer–program interaction along a trajectory leading from purely manual control to control exercised by some compositional algorithm (composing machine). The zone of greatest interest for composition theory is the middle zone of the trajectory, since it allows a great flexibility of approach. The powers of intuition and machine computation may be combined.'

Interfacing Manual and Machine Composition

Unlike human assistants, computers only understand highly formal explanations (i.e. programming code). Formal programming approaches are also necessary for combining manual and machine composition. The notion is illustrated below by examples of common interfaces between manual and machine composition.

Envelopes—functions of time—is an established device for controlling how values change continuously over time. Envelopes are commonly used in sound synthesis, for example, to shape how the loudness or timbre of a single note develops over the duration of this note. In CMask—a composition system for stochastically generating Csound scores—composers can use envelopes to manually shape how the ranges of selected random distributions change over time (Bartetzki, 1997). This approach allows for a very direct control of global parameter characteristics and it is useful, for example, for granular synthesis.

Most musical styles exhibit patterns, and patterns occur on various levels, for example, at the surface (e.g. repetitions or sequences) or at more abstract levels (e.g. a regular metrical structure). The composition system Common Music features rich facilities for composing nested patterns (Taube, 2004). These pattern facilities are particularly expressive, because they afford composers a detailed manual control over aspects such as which patterns are involved, what the parameters of these patterns are, and also how patterns are nested. The Common Music design implements this degree of flexibility by letting composers combine patterns by means of programming.

The composition system OpenMusic proposes the *maquette* concept, which provides composers with high-level control over the musical form (Bresson & Agon, 2008). Using the 'maquette', composers graphically arrange musical segments horizontally along a timeline. These segments can be, for example, composition programs (OpenMusic sub-patches) whose output is then temporally arranged by the 'maquette', like in a MIDI sequencer. Moreover, sub-patches can be defined in such a way that they take further settings in the maquette into account; for example, their horizontal position or input from other sub-patches.

The techniques listed above are all powerful interfaces between manual and computer composition, where composers shape the algorithmic composition process

on a high level of abstraction. All these techniques are deterministic in the sense that they behave predictably; they do not incorporate choices that can be amended later automatically. Deterministic composition techniques share a number of limitations.

Manual composition and deterministic computer composition can only be combined by handing over manual decisions as input to the composition algorithm. It would be very useful if composers could specify, say, that some algorithm generate the pitches of a melody, but allowing for a few important pitches to be composed manually. The above techniques do not support such a direct combination of manual composition and computer composition: the manually specified and the computer-generated values will typically be in conflict. The commonly used algorithmic composition techniques deterministically create a single solution only, and they do not respond to already available partial solutions.

Multiple deterministic algorithms also do not automatically respond to each other. For example, if a composer plans to create the pitches of a piano piece algorithmically, then they have to incorporate all pitch considerations into a single algorithm. It is not possible, for example, to control melodic aspects with one algorithm and the harmony with another—a single algorithm must be designed that either takes care of both aspects or (not uncommonly) simply disregards one of these two aspects. This restriction does not only make the algorithm design more complex, it also greatly impairs the composer's ability to control the compositional results manually. The patterns in Common Music demonstrate the expressive power of manually combining algorithms, and how composers can shape the output with such a technique. Composers would have a far more fine-grained control over the final result if they could combine multiple algorithms that affect the same values (e.g. some algorithms for the melodic and others for the harmonic structure).

Composition systems based on constraint programming do not have these limitations, and they are therefore particularly suited for an approach to CAC that mixes manual compositional decisions and machine composition. Unlike the deterministic approaches of the examples listed above, a constraint-based program starts with a large set of potential solution scores that are then restricted by constraints (compositional rules). The score—either in a simple format resembling a Csound-like (Boulanger, 2000) event list or in a rich and highly nested format—contains variables, which represent information that is unknown at first. For example, all note pitches and durations, the harmonic structure, or the instrumentation might be unknown and represented by unbound variables. These variables are constrained, and a constraint solver algorithm efficiently searches for variable values that satisfy all their constraints (Anders, 2007). Examples of constraint programming systems include: PWConstraints (Laurson, 1996), Situation (Rueda, Lindberg, Laurson, Block & Assayag, 1998) and Strasheela (Anders & Miranda, 2008), to cite but three. We have recently reviewed these and further systems elsewhere (Anders & Miranda, in press).

This paper introduces a number of compositional techniques where manual compositional decisions and machine composition are interwoven. Using such

techniques, composers can inform the computer of their musical intentions; they sculpt the computer composition process. While some of the presented techniques are supported by a number of music constraint systems, other techniques are only supported by our own system, Strasheela: a highly extendable design makes it particularly generic and flexible (Anders, 2007).

Score Information and Variable Domains

All music constraint systems are based on variables with a finite domain of potential values; for example, a set of integers. By setting variable domains, composers specify the intended range of variable values. For example, the choice of allowed note durations affects the resulting rhythmic complexity. Music restricted only to crotchets and quavers differs clearly from music where more duration values including tuplets are permitted. Pitch domains can be customised to specific vocal ranges or to the range of specific musical instruments. The domain of any variable contained in the score can be controlled in this way. For example, a note parameter may specify the instrument of a note and another parameter its articulation. Users can extend the score representation and add further variables.

Variable domains do not need to be uniform; in fact, the domain of each variable can be controlled individually. For example, composers may set different pitch domains for different instrumental parts, or different duration domains for different musical sections. Variable domains can also be sculpted with envelopes. This approach is comparable to dynamically setting boundaries for random values as in CMask (mentioned earlier). However, in a deterministic system like CMask the result of this approach is only controlled with these boundaries (and a random distribution). By contrast, constraint programming allows composers to apply further constraints to the variables whose domains are shaped by envelopes.

The notion of variable domains as a set of possible values allows for a seamless integration of manual and computer composition. By setting a variable domain to only a single value, composers determine this variable manually. For example, composers can set the last pitch of a melody manually, partially write its rhythm and then let the computer fill in the rest.

Analytical information may be represented explicitly in the score. The relation between the note parameters and the analysis is specified by constraints. By setting the domain of analysis variables, composers control the result at a more abstract level. For example, the music may be restricted to a specific scale (e.g. some diatonic scale or a Messiaen mode) by setting the domain of note pitch classes. Microtonal scales are also possible in systems supporting microtonal pitches such as Strasheela.

Some systems optionally support highly abstract analytical information such as the underlying harmonic structure (Anders & Miranda, 2009). By setting variable domains for the harmony, composers can select those chord types that are permitted. For example, the resulting music might be based on triadic harmony, on a vocabulary of Jazz chords, or on a vocabulary of microtonal 7-limit chords (Doty, 2002). It is

also possible to fix manually the root of, say, the first chord; to determine the harmonic rhythm without fixing the harmonic progression itself; or even to fully compose the underlying harmonic structure by hand.

Moreover, composers can control features of the resulting motivic structure manually (Anders, 2009). For example, composers may pre-compose features of the motifs of a piece such as outlining their rhythmic structure (e.g. specifying only the position of the longest note in a motif, or fully stating their rhythm); or sketching their melody (e.g. declaring which tones of the motif are harmonic and which are non-harmonic, specifying a motif's pitch contour (Polansky & Bassein, 1992); or composing the exact interval magnitudes between motif notes). Other variable domains may control the occurrence of motifs in the score. For example, composers can enforce that a particular motif is repeated three times, but leave unspecified the actual look of this motif.

The preceding paragraphs introduced a rich set of possibilities for composers to compose manually in a CAC context—and so far we have only discussed setting variable domains. It is important that such a wealth of options is manageable for users. Ideally, composers should only need to specify the information they want to specify and nothing beyond. It is therefore important for a music constraint system to provide suitable and convenient abstractions for specifying score information—including variable domains.

The Strasheela design proposes a mini language for specifying score information manually (Anders, 2007). Users can use this mini language to specify which score objects they are using, their hierarchic nesting, the domain of variables and other settings. Most settings have a suitable default value, so users only need to specify extra information. There are also convenient shortcuts if settings are shared by a whole sequence of objects.

Figure 1 shows a score specification example in this language. Note that this example uses a pseudo code syntax for simplicity; Strasheela's actual score mini language uses the syntax of the Oz programming language (Anders, 2007). Figure 2 displays this score in music notation. The left-hand side of Figure 2 sketches the specification in music notation, while the right hand side shows a possible solution.

The example in Figure 1 consists of two sequences (seq) that are simultaneous to each other (sim). The first sequence consists of four notes, and the second sequence

```
sim([seq(makeObjects(constructor: makeHarmonyNote
                     n: 4
                     pitch: domain({55, .., 72}) = Notes
         endTime: End)
     seq([chord(duration: 8)
          chord(duration: 16)]
         endTime: End)]
    startTime: 0)
```

Figure 1 Pseudo-code declaration of a partially determined score using Strasheela's score mini language.

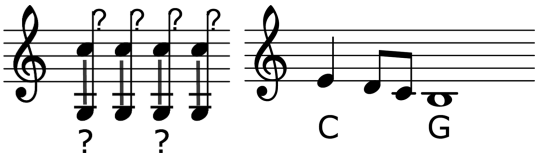

Figure 2 On the left-hand side is a representation of the partially determined score created by the declaration shown in Figure 1. Question marks and vertical bars indicate undetermined information. On the right-hand side is a possible solution.

of two analytical chord objects. Each chord is specified individually, and its duration is fixed (in the present example, the duration 1 indicates a semiquaver figure, so duration 8 is a minim). Other chord parameters are left undetermined. Each chord is therefore indicated with a question mark in the music notation.

For conciseness, the four notes are specified together. Each note is created with the function makeHarmonyNote, and all notes share the same pitch domain (MIDI key-numbers 55–72, 60 is middle C). The music notation displays this pitch domain with a vertical bar. The constructor makeHarmonyNote implicitly applies a number of harmony-related constraints; for example, by default the pitch class of each note is related to its simultaneous chord object. If their parameter harmonicTone is set to true, then the note's pitch class is an element of its related chord pitch class set. Anyway, the specification does not declare which chords are simultaneous to a given note, because the note durations are left undetermined—together with all other note parameters. Flags in the form of question marks indicate the unknown note durations in the notation. Nevertheless, the note sequence and the chord sequence should end at the same time: their end time parameter is set to the same variable End. Note that this variable could be undetermined in principle. The full score starts at score time 0.

Some variable domain settings are done more concisely by constraint applications instead of specifying them in the score mini language. The first and the last note of the present example could be set to harmonic tones with the following two lines of code (the variable Notes was set in Figure 1 to the four notes of the example). The notes in between can be non-harmonic tones (in the solution, the second tone, D, is non-harmonic in C major), as follows:

```
harmonicTone(first(Notes))=true
harmonicTone(last(Notes))=true
```

Constraints

The previous section outlined the range of variables available in music constraint systems. In principle, arbitrary relations between these variables can be enforced by constraints. For example, the repetitive demeanour of minimal music or dodecaphonic music can be expressed by a constraint that enforces a cyclic repetition. Other

examples constrain a rapid passage for harp in such a way that no pedalling is required; specify that certain chords form a cadence; or require long note durations on a downbeat. By defining musical constraints, composers implement the machine composition side of CAC: the computer then automatically searches for a solution that fulfils all constraints.

The following formula shows the implementation of a constraint that models a very simple melodic rule. The interval (the absolute distance) between the notes `N1` and `N2` is constrained not to exceed a fifth (7 semitones). This constraint could be applied, for example, to all pairs of consecutive notes declared in Figure 1.

$$|\texttt{getPitch}(\texttt{N1}) - \texttt{getPitch}(\texttt{N2})| < 8.$$

Highly complex music theories can be modelled with constraints. The constraint above is actually a simplified version of a Palestrina-style counterpoint rule, which is reflected by many counterpoint treatises. In its strict form, it permits intervals between a minor second and a fifth, or an octave. Upwards, also the minor sixth is allowed. Additionally, the rule prohibits all diminished and augmented melodic intervals (Jeppesen, 1939). We surveyed the constraint-based modelling of rhythm, counterpoint, harmony, melody, musical form and instrumentation (Anders & Miranda, in press). We focused on those systems where users can implement their own music theory, such as PWConstraints, Situation and Strasheela.

While the definition of constraints can be challenging, the application of constraints is often straightforward. Selecting, disabling or replacing constraints can therefore be seen as manual compositional decisions. Constraint applications empower composers to make aesthetic decisions at a high level of abstraction. Strasheela predefines many constraints for the rhythmic and harmonic structure, and composers can decide which constraints to use. For example, should the underlying harmonic progression be smooth by connecting consecutive chords by a small voice-leading distance; or should they follow Schoenberg's recommendations for constructing chord progressions (Schoenberg, 1922)?

Constraints can be defined with arguments, similar to functions. For example, the cyclic repetition constraint mentioned above allows composers to specify after how many elements a sequence is repeated. The voice leading distance between chords could be constrained not to exceed some user-specified value.

Constraint applications do not need to be uniform. In Anders and Miranda (2008), we proposed a generic and convenient formalism to control the set of variables affected by a constraint, which can also be a means for controlling musical form. For example, by applying a cadence constraint to the last three chords of a section, formal boundaries can be highlighted. Also, the same constraint can be applied to different sections with different arguments. For example, the rhythm of one section may be constrained to contain an even proportion of long and short notes, while another section is constrained to contain mostly long notes. In a chord progression, an envelope can control how the degree of dissonance of chords changes over time.

In our experience, complex constraint problems are manageable if they are split into sub-problems that can be controlled by arguments individually. Strasheela's hierarchic music representation makes such an approach straightforward: a sub-problem is a music representation segment with implicitly applied constraints. Sub-problems can be defined as functions that expect arguments and return a music representation segment. Examples of this include: a sub-problem for a voice with counterpoint rules implicitly applied, a sequence of analytical chord objects with an implicit theory of harmony, and specific motif definitions, to cite but three. The system predefines such building blocks, but more importantly users can define their own. Strasheela provides convenient constructors for sub-problems; for example, an existing sub-problem can be extended by inheritance. For instance, a counterpoint voice sub-problem can be extended into a motif definition that applies additional constraints, and all counterpoint constraints and arguments are still supported. The expressive power of combining sub-problems using Strasheela's score mini language can be compared with the 'maquette' concept of OpenMusic (mentioned earlier). By controlling the temporal organisation of sub-problems with Strasheela's mini language and by fine-tuning their arguments, composers can operate at high levels of abstraction.

Example

We will now demonstrate the interplay between manual and machine composition using constraint programming with an elaborated real-world example. The composition *Harmony Studies* by the first author has been created with Strasheela. *Harmony Studies* is written in 31-tone equal temperament (31-ET), a temperament very close to quarter-comma meantone (Fokker, 1955). With its pure thirds and sixths, meantone was the backbone of Renaissance and early Baroque music. The extended 31-tone version also approximates 7-limit musical intervals (e.g. the consonant harmonic seventh), which allows for investigations into novel harmonic territories. Formally, the movements of *Harmony Studies* are inspired by the nineteenth-century prelude: they are constructed from a small number of motivic combinations.

This section discusses the composition process of one movement in more detail. This movement features a hierarchically nested musical form. Globally, its form is organised in four sections *A B C A?*, but each section in turn consists of several sub-segments, down to four formal levels, with motifs as the lowest level. This form has been defined with parameterised sub-problems, implemented as functions. For example, a small number of individual motif-functions have been defined: motif variations are created by specifying different argument settings, which control implicit constraints. Other sub-problems combine multiple formal segments into a higher-level segment.

Each movement of *Harmony Studies* deals with the microtonal possibilities of 31-ET in its own right. The harmony of this particular movement is restricted to

specific chord types. Nevertheless, there is no limitation to a specific scale that reduces the set of 31 tones available. Intervals and chords are commonly denoted by their frequency ratios in microtonal music theory—even if these ratios are only approximated by a temperament—because this specification is the only widely agreed and unambiguous naming convention. Section *A* of our movement uses the chords, harmonic seventh (frequency ratios: 4:5:6:7) and sub-harmonic sixth (1/4:1/5:1/6:1/7), but for a clear harmonic contrast section *B* uses the sub-harmonic sixth and the sub-minor seventh chord (12:14:18:21) instead. Note that all these chords constitute highly consonant tetrads that are not available in 12-tone equal temperament (i.e. none of these chords can be played on the piano).

Chords must not randomly follow each other, and the resulting progression should be convincing. This particular movement enforces a constraint that generalises Schoenberg's notion of an *ascending progression*: consecutive chords always share common pitch classes (harmonic band). Additionally, the root of each new chord does not occur in the previous chord (Anders & Miranda, 2009). Schoenberg also refers to these progressions as *strong progressions* because they constitute a clear harmonic change, and at the same time the harmonic band nicely fuses consecutive chords.

Figure 3 shows the underlying harmonic progression of the first two sub-segments of section *A* at the beginning of this movement. Sub-section boundaries are marked with a breath mark. Remember that different enharmonic spellings indicate different pitches in meantone. While the interval *C, E-flat* is the minor third (6/5, 309.68 cent in 31-ET), the interval *C, D-sharp* is the sub-minor third (6/7, 270.97 cent). A quartertone accidental shifts the pitch by a single 31-ET step (38.71 cents). These 'quartertones' sound the same in 31-ET as the pitches notated with double accidentals. For example, *G-half-flat* is the same pitch as *F-double-sharp* (see the second chord in Figure 3). We chose quartertone accidentals for Figure 3 to underline the various occurring interval sizes. In order to assist deciphering the notation, also a harmonic analysis is provided: C harm 7 indicates the harmonic seventh chord over

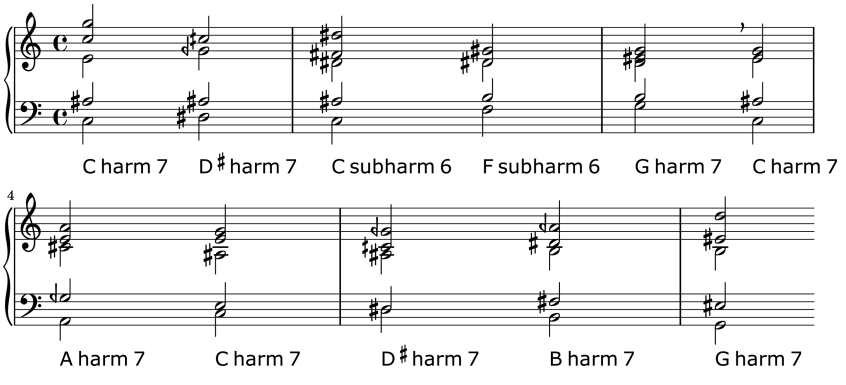

Figure 3 Underlying harmonic progression of the beginning of a movement from *Harmony Studies*, written in 31-tone equal temperament notation (meantone).

C (notated in meantone as *C, E, G, A-sharp*), and subharm 6 is a sub-harmonic sixth chord.

The harmony of these two segments can be perceived as variations of each other. The perception of the musical form has been supported by setting the first and last chords of these segments manually to the tonic and dominant, respectively. Some further constraints were also considered during the composition process, but ultimately not used. For example, we tried to control the harmonic simplicity of the beginning by constraining how often the tonic occurs.

The harmony for other sections of the piece was fine-tuned with further constraints. Figure 4 displays the underlying harmonic progression at the beginning of section *B*. For the sake of an improved harmonic comprehensibility, a cycle pattern constraint was applied to the sequence of intervals between roots and also to the sequence of chord types: only two different intervals (chord types) are alternating. In the solution shown in Figure 4, these two intervals are the diatonic semitone and the fourth upwards (the diatonic semitone and the chromatic '3/4 tone' are enharmonic equivalents in 31-ET). In addition, the harmonic constraints mentioned above are still in force. This combination of constraints resulted in a harmonic sequence that somehow 'wanders around' the circle of 31 fifths.

The actual beginning of the movement is shown in Figure 5. Various manual settings control the motivic organisation of the piece. For example, motif *a* is generated with a homophonic chord progression constructor, which allows for settings such as the number of chords and notes per chord; note parameters such as their pitch and duration domain; and the minimal number of different pitch classes per chord. The sequence of the top notes of the motif chords has been constrained to always descend in pitch. Likewise, the look of motif *b* has been shaped manually by various settings, such as its rhythmic structure and pitch contour. Also, all notes of both motifs have been constrained to harmonic tones, and each instance of motif *a* starts a new harmony.

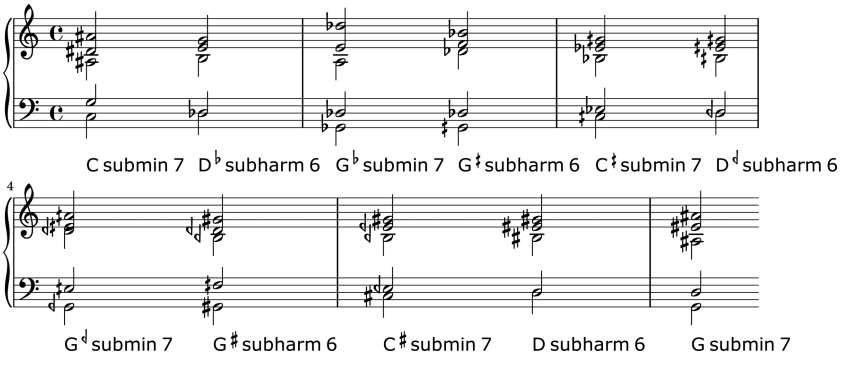

Figure 4 Underlying harmonic progression at the beginning of section *B*, 'wandering around' the circle of 31 fifths, written in 31-tone equal temperament notation.

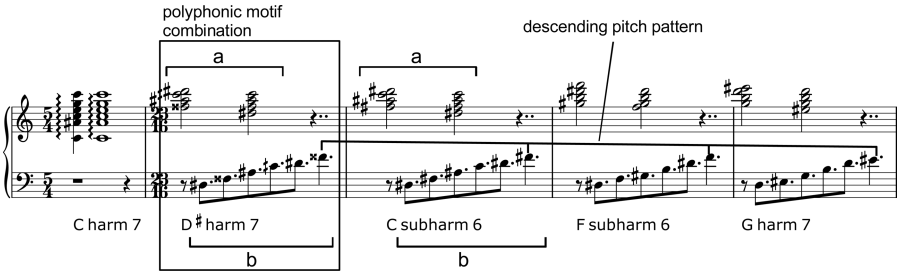

Figure 5 Beginning of this movement from *Harmony Studies*, together with analytical annotations related to constraint applications, written in 31-tone equal temperament notation.

The motifs *a* and *b* have been combined into a polyphonic form segment that is repeated a number of times to form a phrase. This phrase in turn has been fine-tuned by some constraints: the sequence of highest notes of multiple instances of motif *b* descends stepwise. Note that such a constraint also affects the underlying harmony: the harmony cannot be pre-computed in this case, but is always generated together with the motivic structure. Variations of the phrase shown in Figure 5 occur several times in the movement. Besides the evolving underlying harmony, other small changes have been applied. These variations are again controlled manually.

In summary, this example shows how manual and machine composition can work together, even in complex real-world cases. The harmony of this movement is largely computer composed—controlled by a Schoenberg-inspired theory and some manual settings such as the chord types permitted and the harmony at section boundaries. Conversely, the motivic organisation has been composed by hand to a large extent. However, the actual motif pitches depend on the underlying microtonal harmony.

Concluding Discussion

This paper presented a number of techniques that demonstrate how constraint programming allows for an approach to computer-aided composition where manual and machine composition, respectively, are closely interwoven. Using this programming paradigm, all unknown information in the resulting score can be either filled in by the computer or can be overwritten manually.

Users can scale the importance of manual and automatic composition in the design of their programs. If automatic composition is more important, then the design should focus on elaborated constraint definitions. For manual composition, all variables can be set manually anyway. The program design may additionally introduce higher-level controls, for example, by parameterising sub-problems.

Music constraint programming allows for highly complex music theory definitions. For example, a constraint satisfaction problem may contain a fully

fledged theory of harmony as a single component. When manually controlling the composition process it is important to understand—at least to some degree—what you are actually controlling, which can be demanding for complex problems. Following common software engineering principles definitions can be made more comprehensible by, for example, splitting a large definition into suitable sub-definitions; choosing suitable sub-definition arguments; finding meaningful names for sub-definitions, their arguments, constraints and so forth; and documenting their interface.

Although manual and machine composition can be combined rather freely with constraint programming, they must not contradict each other. For example, if the roots of a chord progression are constrained to be all pairwise distinct, and additionally the composer manually fixes two roots to the same value, then this contradiction has no solution. Such problems can be addressed by softening constraints; for example, by declaring a percentage range for the number of elements in a sequence that must hold a specific constraint. In particular, complex music theory definitions are prone to such over-constrained problems where manual compositional decisions are in conflict with the music theory definition, because the composer might have forgotten some dependency. For example, constraints on the harmonic, contrapuntal and motivic structure may affect each other—as the example above demonstrated—which can lead to conflicting settings. Debugging constraint problems can be hard, because it requires analysing the definition in order to find the cause of the conflict. In complex definitions, isolating this cause often works best by reducing the problem; for example, by trying to solve only a musical segment or by disabling a set of constraints.

For future work, using common music notation could greatly facilitate the process of manually specifying score information: reading music notation is usually easier than reading a textual specification of this score. For example, score mini language expressions could be created with a programmable score editor like ENP (see the article by Kuuskankare & Laurson in this issue). Realising this idea might not be as easy as it appears at first, because music notation was not designed to express unknown score information. Its limitations can be seen in Figure 2 (left-hand side). Nevertheless, it might be possible to extend music notation for such purposes, as Figure 2 suggests.

Acknowledgements

This work was developed in the context of the EPSRC-funded project *Learning the Structure of Music* (LeStruM), grant EPD063612-1.

References

Anders, T. (2007). *Composing music by composing rules: Design and usage of a generic music constraint system*. PhD thesis, Queen's University, Belfast.

Anders, T. (2009). A model of musical motifs. In T. Klouche & T. Noll (Eds.), *Mathematics and computation in music. First international conference, MCM 2007*. CCIS 37 (pp. 52–58). Berlin/Heidelberg: Springer.

Anders, T. & Miranda, E. (2008). *Higher-order constraint applicators for music constraint programming*. Paper presented at Proceedings of the 2008 International Computer Music Conference, Belfast, UK.

Anders, T. & Miranda, E. (2009). *A computational model that generalises Schoenberg's guidelines for favourable chord progressions*. Paper presented at the 6th Sound and Music Computing Conference, Porto, Portugal.

Anders, T. & Miranda, E. (in press). Constraint programming systems for modelling music theories and composition. *ACM Computing Surveys*.

Assayag, G. (1998). *Computer assisted composition today*. Paper presented at the First Symposium on Music and Computers, Corfu, Greece.

Bartetzki, A. (1997). Csound score generation and granular synthesis with Cmask. Retrieved June 12, 2009, from http://gigant.kgw.tu-berlin.de/~abart/CMaskPaper/cmask-article.html.

Berg, P. (2009). Composing sound structures with rules. *Contemporary Music Review*, *28*, 75–87.

Boulanger, R. (Ed.). (2000). *The Csound book. Perspectives in software synthesis, sound design, signal processing, and programming.* Cambridge, MA: The MIT Press.

Bresson, J. & Agon, C. (2008). Scores, programs, and time representation: The sheet object in OpenMusic. *Computer Music Journal*, *32*(4), 31–47.

Collins, N. (2008). The analysis of generative music programs. *Organised Sound*, *13*, 237–248.

Collins, K. (2009). An introduction to procedural music in video games. *Contemporary Music Review*, *28*(1), 5–15.

Cope, D. (1996). *Experiments in musical intelligence.* Madison, WI: A-R Editions.

Doty, D. B. (2002). *The Just Intonation primer. An introduction to the theory and practice of Just Intonation* (3rd ed.). San Francisco, CA: Just Intonation Network.

Ebcioglu, K. (1987). *Report on the CHORAL project: An expert system for harmonizing four-part chorales* (Technical Report RC 12628). IBM, Thomas J. Watson Research Center.

Essl, K. (1989). Zufall und Notwendigkeit. Anmerkungen zu Gottfried Michael Koenigs Streichquartett 1959 vor dem Hintergrund seiner kompositionstheoretischen Überlegungen. In H.-K. Metzger & R. Riehn (Eds.), *Gottfried Michael Koenig.* Musik-Konzepte, 66, München: edition text + kritik.

Essl, K. (2000). Lexikon-sonate: An interactive realtime composition for computer-controlled piano. In B. Enders & J. Stange-Elbe (Eds.), *Musik im virtuellen Raum. KlangArt-Kongreß 1997* (pp. 311–328). Osnabrück: Universitätsverlag Rasch.

Fleischer, R. E. & Scott, S. C. (1997). *Rembrandt, Rubens, and the art of their time: Recent perspectives.* University Park, PA: Penn State Press.

Fokker, A. D. (1955). Equal temperament and the thirty-one-keyed organ. *The Scientific Monthly*, *81*(4), 161–166.

Hiller, L. & Isaacson, L. (1993). Musical composition with a high-speed digital computer. In S. M. Schwanauer & D. A. Lewitt (Eds.), *Machine models of music* (pp. 9–21). Cambridge, MA: MIT Press. (Reprint of article in *Journal of Audio Engineering Society*, 1958.).

Jeppesen, K. (1939). *Counterpoint: The polyphonic vocal style of the sixteenth century.* Upper Saddle River, NJ: Prentice-Hall.

Koenig, G. M. (1980). Composition processes. In M. Battier & B. Truax (Eds.), *Computer music.* Ottawa, Canada: Canadian Commission for UNESCO.

Laske, O. (1981). Composition theory in Koenig's Project One and Project Two. *Computer Music Journal*, *5*(4), 54–65.

Laurson, M. (1996). *PATCHWORK: A visual programming language and some musical applications*. PhD Thesis, Sibelius Academy, Helsinki, Finland.

Miranda, E. R. (2001). *Composing music with computers.* Oxford: Elsevier/Focal Press.

Muscutt, K. (2007). Composing with algorithms: An interview with David Cope. *Computer Music Journal*, *31*(3), 10–22.

Polansky, L. & Bassein, R. (1992). Possible and impossible melody: Some formal aspects of contour. *Journal of Music Theory*, *36*, 259–284.

Roads, C. (1996). *The computer music tutorial.* Chapter 18 'Algorithmic Composition Systems' and Chapter 19 'Representation and Strategies for Algorithmic Composition'. Cambridge, MA: The MIT Press.

Rueda, C., Lindberg, M., Laurson, M., Block, G., & Assayag, G. (1998). *Integrating constraint programming in visual musical composition languages*. Paper presented at ECAI 98 Workshop on Constraints for Artistic Applications, Brighton, UK.

Schoenberg, A. (1922). *Harmonielehre* (Revised edition). Wien: Universal Edition. (Trans. by Roy Carter as *Theory of Harmony*. 1978, Berkeley and Los Angeles: University of California Press).

Taube, H. (2004). *Notes from the metalevel.* London: Taylor and Francis.