School of Music & Sonic Arts
Faculty of Arts, Humanities and Social Sciences
Queen's University Belfast

Thesis submission for the degree PhD in Music (Music Technology)

Composing Music by Composing Rules: Design and Usage of a Generic Music Constraint System

Torsten Anders

13th February 2007

Abstract

This research presents the design, usage, and evaluation of a highly generic music constraint system called Strasheela. Strasheela simplifies the definition of musical constraint satisfaction problems (CSP) by predefining building blocks required for such problems. At the same time, Strasheela preserves a high degree of generality and is reasonably efficient.

Strasheela is highly generic, because it is highly programmable. In particular, three fundamental components are more programmable in Strasheela compared with existing music constraint systems: the music representation, the rule application mechanism, and the search process.

Strasheela features an expressive symbolic music representation. This representation supports explicitly storing score information by sets of score objects, their attributes, and their hierarchic nesting. Any information available in the score is accessible from any object in the score and can be used to obtain derived information. The representation is complemented by the notion of variables: score information can be unknown and such information can be constrained.

This research proposes a rule formalism which combines convenience and full user control to express which score variable sets are constrained by a given rule. A rule is a first-class function. A rule application mechanism is a higher-order function. A rule application mechanism traverses the score in order to apply a given rule to variable sets. This text presents rule application mechanisms suitable for a large set of musical CSPs and reproduces important mechanisms of existing systems.

Strasheela is founded on a constraint programming model which makes the search process programmable at a high-level. The Strasheela user can optimise the search for a particular constraint satisfaction problem by programming a distribution strategy (a dynamic variable and value ordering) independent of the problem definition. Special distribution strategies for efficiently solving various musical CSPs – including complex polyphonic problems – are presented.

Acknowledgements

First and foremost I want to thank my supervisors, Michael Alcorn and Christina Anagnostopoulou, for their guidance and support throughout this research study. I am grateful to Michael for giving me the opportunity to work at the Sonic Arts Research Centre. It was a privilege and fun to work in a place with such a stimulating atmosphere. I am indebted to Christina who helped me to clarify the views presented in this thesis.

I am grateful to many people with whom I have discussed by PhD work: Stefan Bilbao, Ludger Brümmer, Darrel Conklin, Mikael Laurson, Camilo Rueda, Örjan Sandred and Alan Smaill. I have greatly benefited from their experience and constructive criticism. Kilian Sprotte was the first who dared to actually use Strasheela.

I would like to thank the Oz community. Denys Duchier answered countless questions I had when I began with this research. Christian Schulte and Raphael Collet helped me to better understand Oz' constraint programming model. Many more members of the Oz mailing list were always willing to help.

I was very happy being surrounded by so many friendly and helpful fellow PhD students: Tom Davis, David Fee, Jason Geistweidt, Katarzyna Glowicka, Banu Günel, Huss Hacıhabiboğlu, Rachel Holstead, Ravi Kuber, Christopher McClelland, Erdem Motuk, Emma Murphy, Damian Ryan, Philip Strain, Evren Tekin, and Henry Vega. I thank the Queen's University for funding my PhD research.

For detailed feedback on drafts of this work I am grateful to Christina Anagnostopoulou, Eoin Brazil, Cornelius Pöpel, and Chris Share. Chris Share also answered many questions I had on English language usage.

Finally, I would like to thank my wife Mechthild for all her support throughout these years.

Contents

1. Introduction

1.1. Computer-Aided Composition

Computers nowadays play an important role in many areas of music composition and its production. In particular, sound synthesis and effects processing plays an important role. By contrast, computer-aided composition utilises the computer in order to create symbolic scores. It focuses on higher-level musical concepts such as notes, chords, motifs, or the musical texture.

In the field of computer-aided composition (CAC, also known as algorithmic composition[1]) composers formalise their musical intentions and implement these formal specifications as computer programs. These programs output music which the composer then uses in the composition process. For writing such programs, composers use either general-purpose programming languages or special composition systems with programming support. Particularly widely-used composition systems include PatchWork [Laurson and Duthen, 1989; Laurson, 1996], the two PatchWork successors OpenMusic [Assayag and Agon, 1996; Assayag et al., 1999] and PWGL [Laurson and Kuuskankare, 2002], and Common Music [Taube, 1991, 2004]. Also the sound synthesis systems SuperCollider [McCartney, 1996, 2002] and Max [Puckette, 1991, 2002] (including its relatives jMax and Pure Data) support CAC very well. Ariza [2006] provides an extensive list of composition systems.

A rich set of composition techniques have been proposed, including mathematical models, models based on transforming existing data, and models which implement already-existing compositional strategies. Examples of mathematical models are stochastic models [Xenakis, 1992] or self-similar systems [Supper, 2001]. The compositional work of Gérard Grisey or Tristan Murail exemplify transformations of spectral analysis data into instrumental music. Koenig's software Project One and Project Two implemented concepts which stem from serial and chance composition [Laske, 1981].

Practitioners of computer-aided composition use various ways of working. In a pragmatic approach which is often applied, the composer delegates certain subtasks of the composition process to some software. The software takes over tedious subtasks, but the composer never surrenders control over the composition process itself. For example, the composer will often manually edit the output of the software and integrate it into the growing piece. Nevertheless, the delegation of composition subtasks to the computer

[1] Ariza [2005] proposes the term *computer-aided algorithmic composition* (CAAC) for this field, but his term has not yet become widely accepted.

often results in a different musical output than purely manually composed music. For example, self-similar systems often lead to highly complex but musically convincing results which are difficult to compose 'by hand'.

Systematic surveys of techniques in CAC – with historical annotations – are provided by Roads [1996], Miranda [2001], and Taube [2004]. Assayag [1998] outlines the history of computer-aided composition. Papadopoulos and Wiggins [1999] offer a systematic overview with a focus on systems based on techniques from Artificial Intelligence.

1.2. Constraint-Based Computer-Aided Composition

Rule-based approaches for CAC emulate the way in which music theories are typically expressed.[2] For centuries, compositional rules have been an established device for expressing explicit compositional knowledge. For instance, rules on how to create an organum (an early polyphonic form) are already described in the treatise Musica enchiriadis, written about 900 (usually attributed to Hucbald of St. Amand).

Because compositional rules are such a long-established concept, rule-based programming approaches have attracted much attention among composers and scholars. This approach allows a user to implement explicit compositional knowledge expressed by compositional rules in a high-level way. A computer program can then use this knowledge when generating music. The usage of rules for computational models of composition will be motivated further in Sec. 3.2.1.

Constraint programming has proven to be a particularly successful programming paradigm for realising ruled-based systems. Many compositional tasks have been addressed by constraint programming. In addition to tasks inspired by traditional music theory such as the generation of harmonic progressions [Pachet and Roy, 2001] or counterpoint [Schottstaedt, 1989; Laurson, 1996], examples include purely rhythmical tasks [Sandred, 2003], Ligeti-like textures [Laurson and Kuuskankare, 2001], or instrumentation [Laurson and Kuuskankare, 2001].

The attraction of constraint programming is easily explained. Constraint programming allows users to model complex problems in a simple way. A problem is stated by a set of *variables* (unknowns) and *constraints* (relations) between these variables. A compositional task is stated by (i) a music representation in which some musical aspects are unknown – and therefore represented by variables – and (ii) compositional rules which impose constraints on these variables. For instance, a chord can be expressed by an event list and the chord pitches can be variables. Some harmonic rules may specify how the chord pitches are related to each other. In the terminology of constraint programming, the modelled problem or task is referred to as a *constraint satisfaction problem* (CSP).

[2] In this text, the term 'rule' (and 'rule-based') primarily denotes the musical concept of a compositional rule. In particular, this term does not implicitly refer to any specific programming technique (e.g. the term does not implicitly refer to a Prolog rule [Bratko, 2001] or a condition-action rule [Russell and Norvig, 2002]). Instead, the text explains how the musical concept is realised as a programming concept in different systems.

In a solution to a CSP, every variable has a value which is consistent with all its constraints. For example, the solution of a musical CSP is a fully-determined score (or score excerpt such as a pure chord progression) consistent with all constraints expressed by the rules. Existing constraint programming systems (abridged: constraint systems) can efficiently solve a CSP – a fact which has greatly contributed to the popularity of constraint programming.

A musical CSP implements a music theory model as a computer program: when the program is executed, it generates music which complies with the modelled theory. The present text defines a music theory model as a model of musical concepts and how these concepts are related to each other. A music theory model implemented by a musical CSP must be fully formalised (e.g. fully expressible in mathematical notation). However, in the context of the present text a music theory model does not necessarily need to be consistent with any existing music. For instance, a composer may develop some musical CSP (and implicitly define a theory model) ad-hoc to generate some subpart of a new composition.

A musical CSP can always be defined 'from scratch' in a general constraint system. For instance, such a CSP can be defined in a regular programming language with support for constraint programming. However, subject-specific CSPs share a considerable amount of subject-specific knowledge: all musical CSPs require modelling of musical knowledge. For instance, concepts such as note, pitch, or voice are required in a large number of musical CSPs. Whenever a musical CSP is defined 'from scratch', all this knowledge must be modelled anew. What's more, any subject-specific optimisation of the search process must also be carried out again (if the chosen constraint system supports such optimisations at all).

Several generic music constraint systems have been proposed. A *generic music constraint system* predefines general musical knowledge and building-blocks shared by many musical CSPs and so highly simplifies the definition of such problems. For example, such a system may provide a specific music representation, templates to simplify the definition of compositional rules, or mechanisms to conveniently control how a rule is applied to the score. Examples of such systems are PWConstraints [Laurson, 1996], and Situation [Rueda et al., 1998; Bonnet and Rueda, 1999]. A pioneering system is Carla [Courtot, 1990]. Further examples include the aggregation of the music representation MusES with the constraint system BackTalk [Pachet and Roy, 1995], OMRC [Sandred, 2000a, 2003][3], Arno [Anders, 2000], OMBacktrack [Truchet, b], and OMClouds [Truchet et al., 2001, 2003]. The number of existing systems underlines the high level of interest in music constraint programming.

The constraint programming paradigm is well-suited to the needs of computer-aided composition. Composers often prefer a way of working which is situated somewhere between composing 'by hand' and formalising the composition process such that it can be

[3]OMRC is defined on top of OMCS, which in turn is a port of the PWConstraints subsystem PMC from its host composition system PatchWork [Laurson, 1996] to the descendant OpenMusic [Assayag et al., 1999].

delegated to the computer (cf. [Laske, 1981]). Constraint programming supports this way of working very well. For example, the composer can determine some aspects of the music (e.g. certain pitches) by hand and constrain other aspects by rules. Alternatively, the composer may specify the high-level structure (e.g. the formal structure) manually and let the computer fill in the details. Furthermore, composers usually do not fully formalise certain aspects of the composition process before they start composing. Instead, the formalisation is often an integral aspect of the composition process itself. A compositional task defined by means of constraint programming can be shaped in a highly flexible way during the composition process by the adding, removing and changing of individual rules.

Established composers have already made extensive use of constraint programming. These include Antoine Bonnet (e.g. for *Épitaphe* for 8 brass instruments, 2 pianos, orchestra and electro-acoustics, 1992–1994, using Situation [Bresson et al., 2005]), Magnus Lindberg (*Engine* for chamber orchestra, 1996, using PWConstraints [Rueda et al., 1998]), Georges Bloch (*Palm Sax* for seven saxophones, using Situation [Rueda et al., 1998]), Örjan Sandred (*Kalejdoskop* for clarinet, viola and piano, 1999, using OMRC, [Sandred, 2003]), Jacopo Baboni Schilingi (*Concubia nocte, in memoria di Luciano Berio* for soprano and live computer, 2003, using OMCS)[4], Johannes Kretz (*second horizon* for piano and orchestra, 2002, using both OMRC and OMCS [Kretz, 2003]), and Hans Tutschku (*Die Süsse unserer traurigen Kindheit*, music theater, 2005, using OMRC)[5].

1.3. Requirements for a Generic Music Constraint System

Generic music constraint systems differ in their degree of generality. A system is more generic if more musical CSPs can be implemented in this system. A *most generic music constraint system* would allow its user to implement any music theory model conceivable. From the perspective of the present research, such a system is an ideal system (as long as efficiency concerns are ignored). For example, composers usually prefer to make compositional decisions themselves; only with reservations do they accept a composition system which restricts their compositional freedom. A most generic music constraint system would not restrict its users to any set of specific CSP classes.

The following paragraphs state a few conditions a most generic music constraint system would have to meet. The present research does not propose such a most generic system for efficiency reasons. Nevertheless, these conditions are important as a guideline for the design of a new system.

A most generic system allows the user to leave arbitrary information about the music unknown – such unknown information is represented by variables which can be freely constrained. For example, the rhythmical structure of the music can be unknown and freely constrained in such a system (e.g. the metric structure or the duration of notes).

[4]Personal communication at PRISMA (Pedagogia e Ricerca sui Sistemi Musicali Assistiti) meeting, January 2004 at Centro Tempo Reale in Florence.

[5]Personal communication at PRISMA neeting, June 2006 at IRCAM, Paris.

Similarly, the musical texture can be constrained (e.g. whether the music is homophonic, polyphonic, or some piano-like texture where the number of voices is constantly changing, etc.). The pitch structure of the music can be unknown and constrained (e.g. the CSP implements a conventional theory of harmony, a dodecaphonic theory, or a microtonal theory of harmony). Also, the instrumentation, or sound synthesis details (e.g. envelopes for various parameters) can be unknown and constrained. A most generic system allows for an extreme case (which is only theoretical), where all information about the music is unknown in the CSP definition. The set of solutions for this extreme CSP contains any conceivable score. The user can freely constrain any unknown information in order to reduce the set of solutions as desired.

A most generic system provides access to arbitrary musical information required for the definition and application of compositional rules in order to support arbitrary musical CSPs. For example, traditional counterpoint rules require detailed information which is deduced from the information explicitly represented in common music notation. For instance, a common contrapuntal rule permits dissonant note pitches in situations where several conditions are met which involve various musical aspects: a note may be dissonant in cases where it is a passing note on an weak beat and below a certain duration. This rule requires harmonic information which is deduced from the pitches of simultaneous notes (whether a certain note is dissonant), melodic information which is deduced from the pitches of notes in the same voice (whether this note is a passing note), metric information which is deduced from the position of the note in a measure (whether this note is on an weak beat), and rhythmical information (the note's duration).

Existing generic music constraint systems, however, are designed to cover specific ranges of musical CSPs. These systems support the formalisation of certain music theory models very well, but other theory models are hard or even impossible to define. For example, OMRC is designed solely for solving rhythmical CSPs, and Situation is best suited to harmonic CSPs. PWConstraints' subsystem score-PMC is designed to solve polyphonic CSPs, but score-PMC requires a fully determined rhythmical structure in the problem definition (i.e. only note pitches can be constrained).

A most generic system not only supports rhythmical, harmonic, as well as polyphonic CSPs etc. Such a system also supports complex CSPs which constrain all these aspects (and more) at the same time. Existing systems are not well-suited for addressing such complex CSPs.

Nevertheless, to a certain extent existing systems are already generic. That is, they support a considerable number of musical CSPs. These systems are generic because the allow the user to program a CSP. In particular, existing systems support the free definition of compositional rules: compositional rules can make use of arbitrarily complex expressions.

Still, only certain aspects of a musical CSP can be programmed in these systems. Other aspects cannot be changed or these systems only offer a limited set of selectable options. For example, the music representations of many existing systems predefine common musical concepts such as notes, pitches and durations which greatly simplifies the definition

of many musical CSPs. However, the user has only limited influence on the form of these representations. Therefore, the representations are only well suited for a limited set of problems. In particular, the representations of existing systems limit the explicit score information that can be stored and what derived information can be accessed. For example, many systems provide a sequential music representation and primarily support deriving information from sets of score object which are positionally related (e.g. neighbouring notes or chords in a sequence). Access to other derived information (e.g. whether a note is on an weak beat, or whether a note is dissonant with respect to the chord expressed by its surrounding notes in a polyphonic texture) is restricted – which clearly affects the set of CSPs which can be defined in these systems.

In addition, the search strategy of existing systems (i.e. how these systems find a solution) is usually optimised for specific classes of musical CSPs. In effect, systems sometimes even purposefully restrict their users to CSPs which they can solve efficiently. For instance, score-PMC does not allow the user to constrain the temporal structure of music, because a determined temporal structure is required by the polyphonic music search approach of score-PMC to compute an efficient static search order [Laurson, 1996]. Similarly, the search strategy of Situation (which performs a consistency enforcing technique to reduce the search space) is optimised for its specific music representation format [Rueda et al., 1998].

The present research proposes a highly generic music constraint system. This system allows the user to define and solve a large set of musical CSPs and is well suited for complex musical CSPs (i.e. CSPs which depend on much musical information). At the same time, this system performs reasonably efficiently – even for problems which were hard or even impossible to solve in previous systems due to their computational complexity (e.g. polyphonic CSPs in which both the rhythmical structure and the pitch structure is constrained). The design of the system is outlined in Sec. 1.4.

This research originates from the field of computer-aided composition. Nevertheless, its results are applicable to areas outside this field. A composer can apply a generic music constraint system as an assistant in the composition process, a music theorist can use it as a testbed to evaluate a music theory, an analyst can conduct a rule-based analysis, and a teacher can demonstrate the effect of different compositional rules to students.

1.4. The Approach Taken

The present research proposes to make music constraint systems more generic by making them more programmable. This proposal is exemplified in the design of the generic music constraint system Strasheela.[6] When comparing Strasheela with previous systems,

[6]Strasheela is also the name of an amicable and stubby scarecrow in the children's novel *The Wizard of the Emerald City* by Alexander Volkov [Wolkow, 1939] in which the Russian author retells *The Wonderful Wizard of Oz* by Baum [1900]. The latter inspired the name for the programming language Oz [van Roy and Haridi, 2004], which forms the foundation for the prototype of the Strasheela

three important aspects are (more) programmable: the music representation, the rule application to the score, and the search strategy.

Strasheela's music representation aims to conveniently provide any information required to express musical CSPs. To this end, the representation is highly extendable. Representation building blocks required for many CSPs are ready-made, but Strasheela additionally predefines building blocks which assist the user in extending the representation according to their needs.

Strasheela defines a novel music representation in the spirit of CHARM (Common Hierarchical Abstract Representation for Music, [Harris et al., 1991]). Two principles have been adopted from CHARM. Like CHARM, Strasheela's representation is based on the notion of data abstraction [Abelson et al., 1985] and it supports user-controlled hierarchic nesting of score objects.

Strasheela's representation complements these principles by other principles drawn from the music representation literature, for example, selectable score parameter (music magnitude) representations [Pope, 1992] (e.g. a pitch can be represented by a key-number, cent or frequency value), bidirectional links between score objects to facilitate free traversing in the score hierarchy [Laurson, 1996], temporal containers which organise their elements sequentially or simultaneously in time [Dannenberg, 1989], organisation of musical data types in a user-extendable class hierarchy [Pope, 1991; Desain and Honing, 1997], and a highly generic data abstraction interface realised by higher-order functions [Desain, 1990].

The Strasheela user freely controls which variables in the music representation are constrained by which compositional rule. To this effect, Strasheela fully decouples the definition and application of a rule in order to make the rule application freely programmable.

Strasheela proposes the encapsulation of compositional rules in functions (actually procedures) as first-class values [Abelson et al., 1985]. This approach allows the user to define rule application mechanisms as higher-order functions expecting rules (i.e. functions) as arguments. Several rule application functions suited to many CSPs have been defined, which either reproduce rule application mechanisms of existing systems or constitute convenient novel application mechanisms. The user can easily define further rule application functions according to their needs (as shown in Sec. 6.2.4).

To be useful in practice, a constraint system must be reasonably efficient. It makes a big difference whether a CSP takes seconds or hours to solve. Strasheela is founded on a constraint programming model based on the notion of computation spaces [Schulte, 2002]. This model makes the search process itself programmable at a high-level. The programmable constraint model allows the user, for example, to optimise the search process for CSPs with a particular structure (e.g. harmonic CSPs or polyphonic CSPs)

composition system.

The scarecrow's brain consists only of bran, pins and needles. Nevertheless, he is a brilliant logician and loves to multiply four figure numbers at night. Little is yet known about his interest in music, but Strasheela is reported to sometimes dance and sing with joy.

by defining what decisions are made during search (the distribution strategy, i.e., the branching heuristics). For instance, the user can control the order in which variables are visited in the search process – depending on the information available at the time of the decision (dynamic variable ordering). This decision order influences immensely the size of the search space. Most previous systems do not allow the user to customise this. In Strasheela, such optimisations are independent of the actual problem definition, which allows the user to easily test a CSP with different search strategies or to reuse proven strategies.

Several novel score distribution strategies have been defined for Strasheela which are suitable for a large range of musical CSPs. In particular, Strasheela provides a score distribution strategy which allows the user to efficiently solve polyphonic CSPs in which both the rhythmical structure as well as other parameters (e.g. pitches) are unknown and constrained in the problem definition [Anders, 2002]. Previous systems discouraged or even disallowed the definition of such problems for efficiency reasons.

1.5. Outline

This thesis consists of four parts: a literature review, a motivation of a generic music constraint system including a description of its requirements, a presentation of a system which fulfills these requirements, and an evaluation of this system.

Literature Review: The related literature is surveyed in two separate chapters, because the presented two research fields have been relatively independent so far. Chapter 2 reviews how music can be represented in computer programs. The chapter discusses both the actual information to express, and fundamental computer science concepts used to represent this information adequately. Chapter 3 surveys the research into constraint-based music composition. After introducing the field in general, the chapter reviews generic music constraint systems which support a considerable number of musical CSPs (the present text continues the research which led to these systems).

Motivation: Chapter 4 analyses the shortcomings of existing generic music constraint systems in terms of generality (i.e. which musical CSPs cannot be solved by these systems and why), formulates the research goal based on this analysis (the development of a more generic system) and outlines how this research goal is addressed by the design of Strasheela. The analysis singles out three problematic aspects in the design of existing systems: their respective music representation, rule formalism and search strategy. Later parts of this text are often organised in accordance with these three aspects.

Strasheela: Chapters 5 to 9 introduce Strasheela in detail. The first three chapters explain Strasheela's design, that is, its music representation (Chap. 5), rule formalism

(Chap. 6) and search approach (Chap. 7). Chapter 8 demonstrates the use of Strasheela by examples and shows how the different parts of Strasheela work together. Chapter 9 introduces a Strasheela prototype and relates it to the model proposed in this text.

Discussion: The last part evaluates and discusses the results of this thesis. Chapter 10 compares Strasheela's approach with existing systems. The chapter investigates by means of general criteria the respects in which Strasheela is more generic than previous systems. Finally, Chap. 11 summarises the main contributions and suggests further developments.

Because this thesis presents a highly interdisciplinary subject, the text explains many terms, even terminology familiar in one of the related areas. In particular, many terms and concepts common in computer science are briefly defined (notational conventions are specified in App. A).