Motivation: Why Strasheela?

Torsten Anders

29th May 2006

Strasheela is a highly generic constraint-based computer aided music composition system. A generic music constraint system makes the definition of a musical constraint satisfation problem (musical CSP, a music theory expressed by means of constraint programming) more easy than it would be in a general constraint system. Therefore, musicians are interested in such a system. At the same time, a generic music constraint system allows to define and solve a large set of musical constraint satisfaction problems. This text motivates such a system and briefly outlines the design of Strasheela.

1 Constraint-Based Computer Aided Composition

Computational models of composition have long attracted musicians and computer scientists alike. Composers like to explore new compositional approaches in order to develop a distinct personal musical language. Music theorists can evaluate an hypothesis by means of a computational model. For computer scientists, modelling music is challenging because the complexity inherent in music calls for the most advanced computer science concepts.

Notably rule-based approaches have alway stimulated interest.[1] For centuries, compositional rules were an established device for expressing compositional knowledge (for example in music education). Many musicians thus feel comfortable with a computational model based on the notion of rules. For example, rule-based approaches attracted much attention among composers, because by defining rules composers can formalise virtually any explicitly available compositional knowledge as a task which the computer can solve automatically.

[1] 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.

Constraint programming has proven a particularly successful programming paradigm to realise ruled-based systems. Many compositional tasks have been addressed by constraint programming. Besides 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], the modelling of non-European music [Chemillier and Truchet, 2001], or instrumentation [Laurson and Kuuskankare, 2001].

The attraction of constraint programming is easily explained. Constraint programming allows to model complex problems a simple way. A problem is stated by a set of *variables* (unknowns) and *constraints* (relations) between these variables. For example, 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).

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 music representation consistent with all constraints expressed by the rules. Existing constraint programming systems (abridged: constraint systems) can efficiently solve a CSP – a fact which greatly contributed to the popularity of constraint programming.

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 such as the Prolog based systems ECLiPSe [Cheadle et al., 2003] or SICTus Prolog [SICS]. 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).

Therefore, a number of 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 that way 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. Particular important 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 constraint system BackTalk [Pa-

chet and Roy, 1995], OMRC [Sandred, 2000, 2003][2], Arno [Anders, 2000], OMBacktrack [Truchet, Truchet], and OMClouds [Truchet et al., 2001, 2003]. Already the number of existing systems demonstrates the high interest in music constraint programming.

The availability of generic music constraint systems makes music constraint programming accessible for a much larger user community. Often, these systems are explicitly tailored for users without any technical background who want to focus on formalising and solving the specific musical tasks they are interested in. A composer can apply such a system as an assistant in the composition process, a music theorist can use it as a testbed to evaluate a music theory, and a teacher can demonstrate the effect of different compositional rules to students.

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 in the middle between composing 'by hand' and formalising the composition process such that it can be delegated to the computer. 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 first fully formalise certain aspects of the composition process before they start actually 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.

A number of well-established composers 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)[3], and Johannes Kretz (*second horizon* for piano and orchestra, 2002, using both OMRC and OMCS [Kretz, 2003]).

[2]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].

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

2 Wanted: A Generic Music Constraint System

A generic music constraint system makes the definition of *musical* CSPs more easy than it would be in a general constraint system. Therefore, musicians are interested in such a system.

At the same time, a generic music constraint system should not restrict its user to specific CSP classes. For instance, composers usually prefer to make compositional decisions themselves without being restricted by some tool such as a composition system what they can express musically. The user of an *ideal* generic music constraint system (in the sense of a most generic system) can formalise any music theory conceivable which can be stated by a set of rules. The system will create music which complies this theory.

Such an ideal system allows to represent arbitrary aspects of the music by variables which can be undetermined and constrained in the problem definition. Example aspects which can be expressed by variables in such a system include the rhythmical structure of the music, its texture (e.g. the number of notes at any time), the pitch structure, instrumentation, or sound synthesis details (e.g. envelopes for various parameters). In an extreme case, the set of solutions for a single CSP contains any conceivable score.

To allow for arbitrary musical CSPs, an ideal system provides access to arbitrary musical information required for the definition and application of compositional rules. For example, traditional counterpoint rules require much information which can only be deduced from the information explicitly represented in the representation traditionally used for contrapuntal composition (i.e. common music notation). For instance, a common contrapuntual rule permits dissonant note pitches in situations where a number of conditions is met which involve various musical aspects: a note may be dissonant in case it is a passing note on an easy beat and below a certain duration. This rule thus requires information on the harmonic aspect deduced from the pitches of simultaneous notes (whether a certain note is dissonant), information on the melodic aspect deduced from the pitches of notes in the same voice (whether this note is a passing note), information on the metric aspect deduced from the position of the note in a measure (whether this note is on an easy 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 cases very well, but other theories are hard or even impossible to define. For example, OMRC is designed solely for solving rhythmical CSPs, and Situation is best suited for harmonic CSPs. PWConstraints's 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).

Existing systems are programming systems and indeed allow a user to express a considerable number of musical CSPs: the user expresses a compositional task in the programming language used by the system. For example, most systems allow the user to freely

define a compositional rule as a modular subprogram which makes use of arbitrarily complex expressions.

Still, only certain aspects of a musical CSP can be programmed. Other aspects can not be changed or the system only offers 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 this representation and in effect the representation is only well suited for a limited set of problems.

In particular, the representations of existing systems limit what explicit score information 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. allow to access neighbouring notes or chords in a sequence). Access to other derived information (e.g. whether a note is on an easy 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 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 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]. Similarily, the search strategy of Situation (which performs a consistency enforcing technique to distinctly 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 to define and solve musical CSPs which were virtually impossible in previous systems. At the same time, this system performs reasonably efficient – even at problems which were hard 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 the subsequent Sec. 3.

A Side Note: Relation to Learning-Based Approaches

Computational models of music composition also often apply an analysis or learning-based approach instead of a rule-based approach. For example, the work of Cope [1991, 1996, 2000] gained particular interest, also because of the musical quality of his results.

Yet, such an approach is best suited to model an existing style of music for which a corpus of examples is available. Composers, however, are usually less interested in style replication but aim to develop their own distinct musical language. The development of

a tool for composers was the original motivation of the present research, which therefore prefers a rule-based approach.

When comparing a rule-based approach with an analysis or learning-based approach, the former expresses explicit musical knowledge (e.g. statements in first-order logic) whereas the musical knowledge for the latter approach is often implicit (e.g. as weights in an artificial neuronal network). However, a learning-based approach can also lead to explicit musical knowledge. For example, [Morales and Morales, 1995] propose a system which automatically creates rules in first-order logic (horn clauses) given a musical example and rule templates. The textbook by Mitchell [1997] introduces learning techniques including the learning of rules.

Rules won by learning can be used in a rule-based system like handwritten rules. Consequently, a generic rule-based system can also be of interest for the community using an analysis or learning-based approach to model music composition.

3 The Approach Taken

The present research proposes to make music constraint systems more generic by making them more progammable. This proposal is exemplified in the design of the generic music constraint system Strasheela.[4] When comparing Strasheela with previous systems, three important aspects in particular are made (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 to extend the representation according needs.

Strasheela defines a novel music representation in the spirit of CHARM [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 allows for user-controlled hierarchic nesting of score objects.

Strasheela's representation complements these principles by other principles learned from the music representation literature, for example, selectable score parameter (music

[4] 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 composition system.

The scarecrow's brain consists only in 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.

magnitude) representations [Pope, 1992] (e.g. a pitch can be represented by a keynumber, 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 an user-extendable class hierarchy [Pope, 1991; Desain and Honing, 1997], and a highly generic data abstraction interface realised by higher-order functions [Desain, 1990].

It is essential for a generic music constraint system that the user freely controls which variables in the music representation are constrained by which compositional rule. Unlike many previous systems, Strasheela fully decouples the definition and application of a rule to make the rule application freely programmable.

Strasheela proposes to encapsulate compositional rules in functions (actually proceduces) as first-class values [Abelson et al., 1985]. This approach allows to define rule application mechanisms as higher-order functions expecting rules (i.e. functions) as arguments. A number of rule application functions suited for 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 such rule application functions according needs.

Finally, a constraint system must be reasonable efficient to be useful in praxis. 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) by defining what decisions are made during search (the distribution strategy, the branching heuristics). For instance, the user can control in which order variables are visited in the search process – depending on the information available at the time of the decision (dynamic variable ordering). These decisions have immense influence on the size of the search space, but previous systems did not allow the user to customise them. These optimisations are independent of the actual problem definition, which allows to easily test a CSP with different search strategies or to reuse proven strategies.

A number of 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 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 disabled the definition of such problems for efficiency reasons.

References

Abelson, H., G. J. Sussman, and J. Sussman (1985). *Structure and Interpretation of Computer Programs*. MIT Press.

Anders, T. (2000). Arno: Constraints Programming in Common Music. In *Proceedings of the 2000 International Computer Music Conference*.

Anders, T. (2002). A wizard's aid: efficient music constraint programming with Oz. In *Proceedings of the 2002 International Computer Music Conference*.

Assayag, G., C. Rueda, M. Laurson, C. Agon, and O. Delerue (1999). Computer Assisted Composition at IRCAM: From PatchWork to Open Music. *Computer Music Journal 23*(3).

Baum, L. F. (1993, orig. 1900). *The Wonderful Wizard of Oz*. Wordsworth.

Bonnet, A. and C. Rueda (1999). *OpenMusic. Situation. version 3* (3rd ed.). Paris: IRCAM.

Bratko, I. (2001). *Prolog. Programming for Artificial Intelligence*. Addison-Wesley. 3rd ed.

Bresson, J., C. Agon, and G. Assayag (2005). OpenMusic 5: A Cross-Platform Release of the Computer-Assisted Composition Environment. In *10th Brazilian Symposium on Computer Music*, Belo Horizonte, Brazil.

Cheadle, A. M., W. Harvey, A. J. Sadler, J. Schimpf, K. Shen, and M. G. Wallace (2003). Eclipse: An introduction. Technical Report IC-PARC-03-1, IC-Parc, Imperial College London.

Chemillier, M. and C. Truchet (2001). Two Musical CSPs. In *Seventh International Conference on Principles and Practice of Constraint Programming, Musical Constraints Workshop*, Paphos, Cyprus.

Cope, D. (1991). *Computers and Musical Style*. Madison, WI: A-R Editions.

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

Cope, D. (2000). *The Algorithmic Composer*. Madison, WI: A-R Editions.

Courtot, F. (1990). A Constraint Based Logic Program for Generating Polyphonies. In *Proceedings of the International Computer Music Conference*, Glasgow.

Dannenberg, R. B. (1989). The Canon Score Language. *Computer Music Journal 13*(1).

Desain, P. (1990). Lisp as a second language: functional aspects. *Perspectives of New Music 28*(1).

Desain, P. and H. Honing (1997). CLOSe to the edge? Advanced object oriented techniques in the representation of musical knowledge. *Journal of New Music Research 2*.

Harris, M., A. Smaill, and G. Wiggins (1991). Representing music symbolically.

Kretz, J. (2003). Continuous Gestures of Structured Material. Experiences in Computer Aided Composition. In *PRISMA 01*. Milano: EuresisEdizioni.

Laurson, M. (1996). *PATCHWORK: A Visual Programming Language and some Musical Applications*. Ph. D. thesis, Sibelius Academy, Helsinki.

Laurson, M. and M. Kuuskankare (2001). A Constraint Based Approach to Musical Textures and Instrumental Writing. In *Seventh International Conference on Principles and Practice of Constraint Programming, Musical Constraints Workshop*, Paphos, Cyprus.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.

Morales, E. and R. Morales (1995). Learning Musical Rules. In *Proceedings of the IJCAI-95 International Workshop on Artificial Intelligence and Music, 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, Montreal, Canada.

Pachet, F. and P. Roy (1995). Mixing Constraints and Objects: a Case Study in Automatic Harmonization. In I. Graham, B. Magnusson, and J.-M. Nerson (Eds.), *Proceedings of TOOLS-Europe'95, Versailles, France*. Prentice-Hall, Hertfordshire, UK.

Pachet, F. and P. Roy (2001). Musical Harmonization with Constraints: A Survey. *Constraints Journal 6*(1).

Pope, S. T. (Ed.) (1991). *The Well-Tempered Object: Musical Applications of Object-Oriented Software Technology*. MIT Press.

Pope, S. T. (1992). The Smoke Music Representation, Description Language, and Interchange Format. In *ICMC 1992*.

Rueda, C., M. Lindberg, M. Laurson, G. Block, and G. Assayag (1998). Integrating Constraint Programming in Visual Musical Composition Languages. In *ECAI 98 Workshop on Constraints for Artistic Applications*, Brighton.

Russell, S. J. and P. Norvig (2002). *Artificial Intelligence: A Modern Approach* (2nd ed.). Prentice Hall.

Sandred, O. (2000). *OMRC 1.1. A library for controlling rhythm by constraints* (2nd ed.). Paris: IRCAM.

Sandred, O. (2003). Searching for a Rhythmical Language. In *PRISMA 01*. Milano: EuresisEdizioni.

Schottstaedt, W. (1989). Automatic Counterpoint. In M. V. Mathews and J. R. Pierce (Eds.), *Current Directions in Computer Music Research*. The MIT Press.

Schulte, C. (2002). *Programming Constraint Services*, Volume 2302 of *Lecture Notes in Artificial Intelligence*. Berlin, Germany: Springer-Verlag.

SICS. SICStus Prolog. `http://www.sics.se/isl/sicstuswww/site/index.html` (accessed 24 May 2006).

Truchet, C. OMBacktrack Tutorial. `http://www.ircam.fr/equipes/repmus/OpenMusic/Documentation/OMUserDocumentation/DocFiles/Reference/backtracktutorial/` (accessed 25 May 2006).

Truchet, C., C. Agon, and P. Codognet (2001). A Constraint Programming System for Music Composition, Preliminary Results. In *Seventh International Conference on Principles and Practice of Constraint Programming, Musical Constraints Workshop*, Paphos, Cyprus.

Truchet, C., G. Assayag, and P. Codognet (2003). OMClouds, a heuristic solver for musical constraints. In *MIC2003: The Fifth Metaheuristics International Conference*, Kyoto, Japan.

van Roy, P. and S. Haridi (2004). *Concepts, Techniques, and Models of Computer Programming*. MIT Press.

Wolkow, A. (2005, Russian orig. 1939). *Der Zauberer der Smaragdenstadt*. Leiv Buchhandels- und Verlagsanstalt.