Interactive Intelligent Systems Workshop: Music Constraint Programming (3)

Torsten Anders
Interdisciplinary Centre for Computer Music Research (ICCMR)
University of Plymouth
http://cmr.soc.plymouth.ac.uk/

26 November 2007
Outline

1. Introduction
2. Why a Generic Music Constraint System?
3. Strasheela: a Generic Music Constraint System
4. Summary
Recap: Music is Multi-Contextual

- Music forms a complex net of relations between score objects.
- Even single events and their parameters are part of many score contexts (e.g. rhythmical, harmonic, contrapuntal, and formal score contexts).
- Example: some pitch contexts of a single note (marked red).

![Musical staff with annotations]

- Melodic motif
- Auxiliary note phrase
- Notes expressing a harmony

Andante grazioso. ($\frac{\text{d}}{\text{d}} = 120$)
Rules and Constraint Programming

- Music theory describes relations within/between multiple musical contexts by modular rules
- Multi-contextual rule sets hard to model by procedural programming (most often used programming paradigm in CAC)
- Constraint programming well suited for implementing multi-contextual rule sets
What is Constraint Programming?

**Definitions**

A *constraint satisfaction problem* (CSP) states *constrains* (mathematical relations) between *variables* (unknowns) with a specific *domain* (a set of possible variable values). Constrain specify properties of a solution, which is found by search.

**CSP example**

\[ X + Y = 7 \]
\[ \land X < Y \]
where \( X \in \{1, \ldots, 10\} \land Y \in \{1, \ldots, 10\} \)

possible solution: \( X = 3, Y = 4 \)
Advantages of Constraint Programming

- **Complex problems simple to model**: model states only *what*, not *how*
- **There exist efficient solvers** to search for solution(s)
Music Constraint Programming

**Definition**

A *musical constraint satisfaction problem* is a set of rules (constraints) applied to a music representation where some aspects are unknown (variables).

- **Executable implementation of a music theory model**: the user states a music theory and the computer generates music which complies with this theory
- **Declarative definition**: similar to the way music theory is traditionally expressed: by a set of rules (although a musical CSP is more formal)
- **Beneficial for various fields**: composer, music theorist, teacher
- **Challenging for computer scientist and mathematician**
Examples of Musical Constraint Satisfaction Problems

Various music theory areas have been addressed with constraint programming

- Rhythm
- Counterpoint
- Harmony
- Musical form
- Instrumentation

Examples: http://strasheela.sourceforge.net/strasheela/doc/StrasheelaExamples.html
Two Approaches for Defining Musical CSPs

### Implementation ‘from scratch’

- **Examples**: Bach Chorales [Ebcioglu, 1984], Fuxian Counterpoint [Schottstaedt, 1989]
  - **的优点**: Generic approach
  - **缺点**: Requires much work and expertise

### Generic music constraint systems

- **Examples**: PWConstraints [Laurson, 1996], Situation [Rueda, 1998]
  - **优点**: Simplifies definition with predefined building blocks
  - **缺点**: Limits expressiveness
## Two Approaches for Defining Musical CSPs

### Implementation ‘from scratch’

<table>
<thead>
<tr>
<th>Examples</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Bach Chorales</td>
<td>[Ebcioglu, 1984]</td>
</tr>
<tr>
<td>Fuxian Counterpoint</td>
<td>[Schottstaedt, 1989]</td>
</tr>
</tbody>
</table>

- **⊕ Generic** approach
- **⊖** Requires much *work* and expertise

### Generic music constraint systems

<table>
<thead>
<tr>
<th>Examples</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PWConstraints</td>
<td>[Laurson, 1996]</td>
</tr>
<tr>
<td>Situation</td>
<td>[Rueda, 1998]</td>
</tr>
</tbody>
</table>

- **⊕** Simplifies definition with predefined building blocks
- **⊖** Limits expressiveness
Existing Generic Music Constraint Systems

- **Carla** [Courtot, 1992]: pioneering system
- **MusES with BackTalk** [Pachet and Roy, 1995]: expressive music representation plus constraint solver
- **PWConstraints** [Laurson, 1996]:
  - PMC: language for constraining lists of integers, user controlled rule scope with pattern matching mechanism
  - score-PMC: polyphonic CSPs
- **Situation** [Bonnet and Rueda, 1999]: constraining lists of Situation score objects, special support for harmonic CSPs
- **OMRC** [Sandred, 2000]: rhythm constraints
- **Arno** [Anders, 2000]: constraining scores of Common Music [Taube, 1991]
- **OMClouds** [Truchet, 2001]: heuristic constraints on number lists, visual programming system
Motivation for a More Generic System I

Research goal

A more generic music constraint system

- Compared with implementation ‘from scratch’: simplify musical CSP definition: predefined building blocks
- Compared with existing generic systems: highly generic system: allow for large range of musical CSPs
- Reasonable efficient system for various CSPs (not just optimised for specific class of CSPs, as existing generic systems)
Motivation for a More Generic System II

Lessons learnt

Make building blocks **programmable**

- Highly extensible music representation
- Convenient and user-programmable constraint applicators
- Search: user-programmable dynamic variable/value ordering
### Strasheela Usage

- **User** creates **score**
Strasheela Usage

score contains variables (unkowns)

user

score

X

Y

Z
Strasheela Usage

user creates rule

rule creates score
Strasheela Usage

user explicitly applies rules to score
(e.g. harmonic rule to simultaneous notes)
Strasheela Usage

rules constrain variables in score
Strasheela Usage

Introduction
Why a Generic Music Constraint System?
Strasheela: a Generic Music Constraint System
Summary

Strasheela Overview
Music Representation
Rule Formalism

Strasheela Usage

musical CSP

rule

rule applicator

score

solver

solution

user

Torsten Anders

Music Constraint Programming (3)
Strasheela Usage

- User interacts with the system
- Musical CSP
  - Rule applicator
  - Rule
  - Score
- Solver
  - Solution
- Output
  - Music notation
  - Sound synthesis
  - CAC systems
Strasheela Usage

- user evaluates output and often edits CSP

- Musical CSP
  - rule applicator
  - rule
  - score

- Output
  - music notation
  - sound synthesis
  - CAC systems

- User interacts with CSP through:
  - Solver
  - Solution

- User evaluates output and often edits CSP.
Music Representation Principles

Basic principle: **user controls which score information is represented**

- Score consists of score objects
- Explicitly stored information: object type, object attributes, hierarchic nesting of objects
- An object **attribute can be a constrained variable**
- Access to score information via rich interface (incrementally defined: class hierarchy)
- Any information (e.g. score context) accessible from any score object (bi-directionally linked hierarchy)
Strasheela Class Hierarchy

Strasheela music representation class hierarchy (temporal classes)

- **TimeMixIn**
  - startTime
  - endTime
  - duration
  - offsetTime

- **ScoreObject**
  - info

- **Item**
  - containers
  - parameters

- **Parameter**
  - item
  - value
  - unit

- **Container**
  - items

- **Element**
  - **TemporalElement**
    - **Pause**
    - **Event**
      - **Note**
        - pitch
        - amplitude

- **TemporalContainer**
  - Sequential
  - Simultaneous
**Example: A Single Determined Note (1)**

The textual representation of a single note and this note in common music notation

\[
\text{makeScore} \left( \text{note} \left( \text{info: testNote,} \right.ight. \\
\text{startTime: 0,} \\
\text{duration: 2,} \\
\text{timeUnit: seconds,} \\
\text{pitch: 6050,} \\
\text{pitchUnit: midicent}) \right)
\]
Example: A Single Determined Note (1)

The textual representation of a single note and this note in common music notation

```
makeScore(note (info: testNote,
        startTime: 0,
        duration: 2,
        timeUnit: seconds,
        pitch: 6050,
        pitchUnit: midicent))
```
Example: A Single Determined Note (1)

The textual representation of a single note and this note in common music notation

```plaintext
makeScore(note (info: testNote,
            startTime: 0,
            duration: 2,
            timeUnit: seconds,
            pitch: 6050,
            pitchUnit: midicent))
```

\[
\text{\text{\textbackslash m}a\text{\textbackslash k}_{\text{S}c\text{\textbackslash o}\text{\textbackslash r}}(\text{\textbackslash n}o\text{\textbackslash t} (\text{\textit{i}n\text{\textbackslash f}o: \textit{t}e\text{\textbackslash s}t\text{\textbackslash n}e\text{\textbackslash t}N\text{\textbackslash o}t, \text{\textit{s}t\text{\textbackslash a}r\text{\textbackslash t\textbackslash i}t\text{\textbackslash m}\text{\textbackslash e}\text{\textbackslash t}: 0, \text{\textit{d}\text{\textit{u}\textit{r}\textit{a}\textit{t}\textit{i}o}n: 2, \text{\textit{t}\text{\textit{i}m\textit{e}U\textit{n}\textit{i}t: \textit{s}e\text{\textit{c}o\textit{n}d\textit{s}}, \text{\textit{p}i\textit{t}c\textit{h}: 6050, \text{\textit{p}i\textit{t}c\textit{hU}n\textit{i}t: m\text{\textit{i}d\textit{ic}e\textit{n}t}))
```

\[
\text{\textbackslash m}a\text{\textbackslash k}_{\text{S}c\text{\textbackslash o}\text{\textbackslash r}}(\text{\textbackslash n}o\text{\textbackslash t} (\text{\textit{i}n\text{\textbackslash f}o: \textit{t}e\text{\textbackslash s}t\text{\textbackslash n}e\text{\textbackslash t}N\text{\textbackslash o}t, \text{\textit{s}t\text{\textbackslash a}r\text{\textbackslash t\textbackslash i}t\text{\textbackslash m}\text{\textbackslash e}\text{\textbackslash t}: 0, \text{\textit{d}\text{\textit{u}\textit{r}\textit{a}\textit{t}\textit{i}o}n: 2, \text{\textit{t}\text{\textit{i}m\textit{e}U\textit{n}\textit{i}t: \textit{s}e\text{\textit{c}o\textit{n}d\textit{s}}, \text{\textit{p}i\textit{t}c\textit{h}: 6050, \text{\textit{p}i\textit{t}c\textit{hU}n\textit{i}t: m\text{\textit{i}d\textit{ic}e\textit{n}t}))
```
Example: A Single Determined Note (1)

The textual representation of a single note and this note in common music notation

\[
\text{makeScore} (\text{note} (\text{info}: \text{testNote}, \newline \text{startTime}: 0, \newline \text{duration}: 2, \newline \text{timeUnit}: \text{seconds}, \newline \text{pitch}: 6050, \newline \text{pitchUnit}: \text{midicent})))
\]
Example: A Single Determined Note (2)

The hierarchic structure of a single note and its contained parameters (UML)

- :Note
- :Parameter

Bidirectional link: An Item references its Parameters and vice versa

Note Parameters are offsetTime, startTime, duration, ...
Example: A Note Sequence (1)

The textual representation of a container with several notes

```makeScore(sequential (info: myVoice,
  items: [note(duration: 2, pitch: 60),
          note(duration: 1, pitch: 62),
          note(duration: 1, pitch: 64),
          note(duration: 4, pitch: 65)],
  startTime: 0,
  timeUnit: beats(4))))```
Example: A Note Sequence (1)

The textual representation of a container with several notes:

```
makeScore(sequential (info: myVoice,
    items: [note(duration: 2, pitch: 60),
              note(duration: 1, pitch: 62),
              note(duration: 1, pitch: 64),
              note(duration: 4, pitch: 65)],
    startTime: 0,
    timeUnit: beats(4)))
```
Example: A Note Sequence (1)

The textual representation of a container with several notes

```makeScore(sequential (info: myVoice,
    items: [note(duration: 2, pitch: 60),
            note(duration: 1, pitch: 62),
            note(duration: 1, pitch: 64),
            note(duration: 4, pitch: 65)],
    startTime: 0,
    timeUnit: beats(4)))```
Example: A Note Sequence (2)

A container with several notes in common music notation
Example: A Note Sequence (3)

The hierarchic structure of a container with several contained notes

Bidirectional links: A Container references its contained Items and vice versa

Sequential and Note Parameters are omitted for brevity
Example: An Hierarchic Representation with Undetermined Pitches (1)

The hierarchic structure of a polyphonic example (represented by a tree-like nesting of containers and elements)
Example: An Hierarchic Representation with Undetermined Pitches (2)

A polyphonic example where the note pitches are undetermined
Example: An Hierarchic Representation with Undetermined Pitches (3)

The textual representation of a polyphonic example (parts of the score are not specified explicitly but created by function calls)

```plaintext
let makeNotes(n) :=
    collectN(n, f : f() := note(duration: 1,
        pitch: △{60, . . . , 72}))

in makeScore(sim (items: [seq (info: alto,
        items: makeNotes(3)),
        seq (info: tenor,
        items: makeNotes(3))],
        startTime: 0,
        timeUnit: beats))
```
Example: An Hierarchic Representation with Undetermined Pitches (3)

The textual representation of a polyphonic example (parts of the score are not specified explicitly but created by function calls)

```plaintext
let makeNotes(n) :=
    collectN(n, f : f() := note(duration: 1,
        pitch: √{60, ..., 72}))

in makeScore(sim (items: [seq (info: alto,
        items: makeNotes(3)),
        seq (info: tenor,
            items: makeNotes(3))],
            startTime: 0,
            timeUnit: beats))
```
Example: An Hierarchic Representation with Undetermined Pitches (3)

The textual representation of a polyphonic example (parts of the score are not specified explicitly but created by function calls)

```
let makeNotes(n) :=
  collectN(n, f : f() := note(duration: 1,
    pitch: \{60, \ldots, 72\}))

in makeScore(sim(items: [seq(info: alto,
    items: makeNotes(3)),
  seq(info: tenor,
    items: makeNotes(3))],
  startTime: 0,
  timeUnit: beats))
```
Rule Definition

Rule definition (simplified)

A rule is a function expecting variables or score objects as arguments and returning a boolean variable.

Rule example

\[ \text{transpose} : \text{FD int} \times \text{FD int} \times \text{FD int} \mapsto 0 \lor 1 \]

\[ \text{transpose}(\text{Pitch}_1, \text{Distance}, \text{Pitch}_2) := \text{Distance} = |\text{Pitch}_1 - \text{Pitch}_2| \]
Advantages of Rules as Functions

- User can freely define rules on variables and score objects
- User can freely combine rules (e.g., rule *transposition* can be used in rule restricting melodic intervals)
Existing Rule Application Mechanisms

- Already existing systems allow the user to freely define rules
- Tricky part: **how to apply rule to score?**
- Existing systems propose various solutions
  - Only a single (complex) rule is defined and implicitly applied to whole score
  - Index-based rule application
  - Rule application with pattern matching
- Those solutions are limited!
User-Defined Rule Applicators

Rule and rule applicator definition (revised)

- A rule is a first-class function (actually, a first-class procedure)
- A rule applicator is a higher-order procedure: it applies a given rule to variable sets in the score
Example: rule application to hierarchically related score objects

Example: apply melodic rule to pairs of neighbouring notes

Rule application to pairs of neighbouring notes

\[ \bigwedge \text{map2Neighbours(getPitches(myVoice), myRule)} \]

Definition of rule and applicator

\[
\text{myRule}(Pitch_1, Pitch_2) := 7 \geq |Pitch_1 - Pitch_2|
\]

\[
\text{map2Neighbours(xs, fn)} := \text{zip(butLast(xs), tail(xs), fn)}
\]
Example: rule application to selected objects in score hierarchy

Every note in \textit{myScore} is constrained to a diatonic pitch

\[
\bigwedge \text{map}( \text{filter}(\text{collect}(\text{myScore}), \text{isNote}), \text{hasDiatonicPitch})
\]
Advantages of User-Defined Rule Applicators

- Convenient application of rule to specific score context
- Fully user programmable
- Strasheela predefines applicators suitable for various CSPs
Constraining Inaccessible Score Contexts

Problem
Score contexts can be inaccessible in CSP definition due to undetermined variables. Example: if rhythmic structure is undetermined, context of simultaneous notes is inaccessible.

Techniques
- Delayed rule application
- Reified constraints (logical connective constraints)

Example using implication

\[ \text{isSimultaneous}(\text{note}_1, \text{note}_2) \Rightarrow \text{isConsonant}(\text{note}_1, \text{note}_2) \]
Summary

- Constraint Programming Introduction
- Music Constraint Programming Introduction
- Motivation of a Generic Music Constraint System
- Strasheela Introduction
  - Music Representation
  - Rule Formalism