Interactive Intelligent Systems Workshop:
Music Constraint Programming (2)
Music Representation

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

21 November 2007
What is a Music Representation?

- A music representation exhibits musical information, often in a format designed for this purpose.
- Example: common music notation (printed score)
- We are interested in symbolic representations for computer programs (e.g., score data structures)
What is a Music Representation?

- A music representation exhibits musical information, often in a format designed for this purpose.
- Example: common music notation (printed score).
- We are interested in symbolic representations for computer programs (e.g., score data structures).
Outline

1. Introduction
2. Fundamental Music Representation Concepts
3. Programming Concepts in a Music Representation
4. Summary
**Event-List: a Basic Representation**

**Definitions**

An *event* is score object which produces sound. An event features a set of parameters.

A *parameter* is a basic magnitude in a music representation. Examples: start time, duration and pitch.

**Event-list example**

\[
\text{[note(start:0, dur:1, pitch:60, amp:0.5),}
\text{ note(start:1, dur:1, pitch:64, amp:0.5),}
\text{ note(start:2, dur:1, pitch:67, amp:0.5)]}
\]
Parameter Representation

- Single parameters can be represented in various ways
- Different representations are suitable for different purposes
Parameter Representation Examples I

**Example: typical pitch representations**

- As frequency in Hertz – unambiguous representation
- As key-number (MIDI number) – more higher level (depends on concert pitch)
  - As integer
  - As float – for microtonal representation
- As symbolic note number (e.g. $g^\#4$) – representation with harmonic information
Parameter Representation Examples II

**Example: temporal parameter representations**

- Measured in seconds – unambiguous representation
- Measured in beats – more higher level (depends on tempo, which may vary)
Parameter Representation Examples III

Example: composite parameter representation

CHARM represents a pitch by the following features:

\[ \text{noteName} \times \text{accidental} \times \text{octave} \]

Example: \( \text{pitch}(\text{noteName}: d, \text{accidental}: b, \text{octave}: 4) \)

Advantages: more information represented than in key-number (enharmonic representation), independent aspects of the pitch are directly accessible
Why Hierarchic Representations?

- Music consists of groups of score objects (e.g., chords, scales, motifs, voices)
- Those groups of score objects can even be hierarchically nested (e.g., Schenkerian analysis, hierarchic grouping structure)
- Hierarchic representations group score objects by nesting them in containers.
A Musical Example

Béla Bartók. Mikrokosmos, No. 87, beginning

In the following, this example is expressed by different hierarchic representations
Representation with Specialised Containers in a Tree

- Widely used for music notation languages (e.g., Lilypond)
- Also used in composition systems (e.g., PWGL)
Nested Temporal Containers (1)

- General means to express the hierarchic structure of a set of score objects
- A representation using generic containers can substitute a tree of specialised containers
- Used in various systems for sound synthesis (e.g., Nyquist) and music composition (e.g., Haskore)

Representation of the Mikrokosmos example by tree of generic temporal containers
Nested Temporal Containers (2)

Generic temporal containers can express different hierarchic nestings which represents different musical information (e.g., part structure vs. measure structure).

Representation of the Mikrokosmos example by an alternative tree of generic temporal containers.

- Sequential [score]
  - Simultaneous [measure]
    - Sequential [part]
      - Note
    - Sequential [part]
      - Pause
  - Simultaneous [measure]
    - Sequential [part]
      - Note
    - Sequential [part]
      - Simultaneous [chord]
Containers in an Acyclic Graph

- Score objects often belong to multiple score contexts at the same time.
- This representation can subsume in a single representation the information expressed by different hierarchic nestings of temporal containers.
- Used in composition and analysis systems (e.g., CHARM).

Representation of the Mikrokosmos example by a music representation consisting of multiple orthogonal containers which form an acyclic graph.
Why Using Programming Concepts in a Music Representation?

- For many purposes (e.g., music composition), a music representation cannot be fixed – it must grow with its use.
- Example: composers are constantly developing novel composition and instrument playing techniques.
- Integrating a music representation into a programming language:
  - Particular flexible representation
  - Still comprehensible (e.g. extensions can be modular)
Why Using Programming Concepts in a Music Representation?

- For many purposes (e.g., music composition), a music representation cannot be fixed – it must grow with its use.
- Example: composers are constantly developing novel composition and instrument playing techniques.
- Integrating a music representation into a programming language:
  - Particular flexible representation
  - Still comprehensible (e.g. extensions can be modular)
Why Using Programming Concepts in a Music Representation?

- For many purposes (e.g., music composition), a music representation cannot be fixed – it must grow with its use.
- Example: composers are constantly developing novel composition and instrument playing techniques.
- Integrating a music representation into a programming language:
  - Particular flexible representation
  - Still comprehensible (e.g. extensions can be modular)
Problem: Flexible Data Access Required

Problem to solve

We often want to represent the same musical information in different ways for different purposes. Also, the user should decide in which format the information is stored.

Example: For music notation output, we need symbolic and tempo-dependent note durations, and for playback we need note durations in seconds. We only want to store a single note duration value in a user-chosen format.

Question

What computer science concept do you suggest to use in this case?
Problem: Flexible Data Access Required

Problem to solve

We often want to represent the same musical information in different ways for different purposes. Also, the user should decide in which format the information is stored.

Example: For music notation output, we need symbolic and tempo-dependent note durations, and for playback we need note durations in seconds. We only want to store a single note duration value in a user-chosen format.

Question

What computer science concept do you suggest to use in this case?
Data Abstraction Concept

Data abstraction concept

- An abstract data type (ADT) *encapsulates* a set of internal values
- Set of operations defined for this ADT (the *interface*)
- Only interface operations can process datum (do direct access to internal values)

Typical interface operations

- constructor, accessors (also known as selectors) and modifiers (also known as mutators or setters) to access and change the encapsulated data,
- operators for type-checking, equality checking, and special purpose operators
Data Abstraction Concept

Data abstraction concept

- An abstract data type (ADT) *encapsulates* a set of internal values
- Set of operations defined for this ADT (the *interface*)
- Only interface operations can process datum (do direct access to internal values)

Typical interface operations

constructor, accessors (also known as selectors) and modifiers (also known as mutators or setters) to access and change the encapsulated data, operators for type-checking, equality checking, and special purpose operators
Data Abstraction Example

Example interface function

```plaintext
getDuration(myNote, 'beats')
getDuration(myNote, 'seconds')
```

Example systems

**CHARM** defines special arithmetic operations for parameters such as time (i.e. a time point) and duration (i.e. an interval between time points): adding a time and a duration results in a time, whereas adding two durations results in a duration.

**MusES** introduces rich algebra for enharmonic pitches: e.g. diminished fifth of c is g♭ and the augmented fourth of c is f♯ (both the same key on the piano) – an important distinction for harmonic processing.
Problem: How to Avoid Redefinition?

Problem to solve

A score contains data entities of many different data types.

Examples:

- Music notation: notes marking pitch and timing information, various articulation and expression signs, staves to organise notes in voices, clefs, key and meter signatures etc.
- Analytical information: representation of harmonic or motific structure

Question

How to avoid defining an enormous set of different types in an unrelated way?
How to make type set easily extendable by user?
Problem: How to Avoid Redefinition?

Problem to solve

A score contains data entities of many different data types.

Examples:

- Music notation: notes marking pitch and timing information, various articulation and expression signs, staves to organise notes in voices, clefs, key and meter signatures etc.

- Analytical information: representation of harmonic or motivic structure

Question

How to avoid defining an enormous set of different types in an unrelated way?

How to make type set easily extendable by user?
Object-oriented programming (OOP) is a form of data abstraction.

A *class* definition is quasi a data type definition, and an actual datum with that "type" is an *object* of that class (also called a *class instance*).

**Example:** a class *note* defines a note "type", a score then contains many note objects (instances of the class *note*).
Object-oriented programming extends notion of data abstraction

*Inheritance* makes incrementally defined abstract data types possible: shared internal values (*attributes*) and operations (*methods*) are defined only once in a more general superclass, from which a subclass *inherits*.

**Example:** note and pause objects have much in common (for example temporal parameters such as a start time and a duration): they may have a common superclass which defines their shared aspects.
Score Class Hierarchy Example

A more complex example (class hierarchy of the composition system Common Music [Taube, 1993])

- Object
  - Event
    - Note
      - Midi-Note
    - Rest
      - FM-Violin
      - [...]
  - Collection
    - Thread
    - Merge
    - Algorithm
      - [...]

Problem: How to Abstract Complex Queries

Problem to solve

We often want to query complex information about a hierarchically nested score. The information can be collected by tree or graph traversal.

Examples

- Collect the pitches of all notes in the score (e.g., for some analysis processing)
- Transform a hierarchically nested score into a flat event list (for sound output)

Question

How can we avoid to always write a tree traversal from scratch whenever we need a new query operation?
Problem: How to Abstract Complex Queries

Problem to solve

We often want to query complex information about a hierarchically nested score. The information can be collected by tree or graph traversal.

Examples

- Collect the pitches of all notes in the score (e.g., for some analysis processing)
- Transform a hierarchically nested score into a flat event list (for sound output)

Question

How can we avoid to always write a tree traversal from scratch whenever we need a new query operation?
Higher-Order Programming Concept

First-class functions

Functional programming makes operations (functions) first-class values. A first-class value can be

- stored in a data structure
- input into a function as argument
- returned from a function as result

A function expecting other functions as argument or returning functions is called a *higher-order* function.
Higher-Order Programming Examples I

\textit{filter} returns list elements for which a given function returns true

\begin{align*}
\text{filter}([1, 2, 3, 4, 5], \text{isEven}) &= [2, 4]
\end{align*}
Higher-Order Programming Examples II

*sort* sorts a list according to a given comparison function

\[
\text{sort}([4, 1, 3, 5, 2], f : f(x, y) := x < y) = [1, 2, 3, 4, 5]
\]
Higher-Order Programming Examples III

Extracting an event-list with only specific events (events starting after 60 seconds) from an hierarchically structured score (set-builder notation)

\[ \{ x : x \text{ contained in } myHierarchicScore \wedge isEvent(x) \wedge \text{getStartTime}(x) > 60 \} \]
Higher-Order Programming Examples IV

Extracting an event-list with only specific events with the function \texttt{filter}

\begin{verbatim}
let minTime := 60
in filter( collectAllObjects(myHierarchicScore),
           f : f(x) := isEvent(x) \land getStartTime(x) > minTime)
\end{verbatim}
Higher-Order Function Definition

Higher-order functions (like \textit{filter}) are defined like any other function

\[
\text{filter}(xs, test) := \begin{cases} 
\text{if } \text{isNil}(xs) \text{ then nil} & \text{/* abort condition} \\
\text{/* Apply the function test to first element of the list xs. Collect only matching elements, but recur in any case with tail of the list xs} \\
\text{else if } test(\text{head}(xs)) \\
\text{then cons} (\text{head}(xs), \\
\text{filter}(\text{tail}(xs), test)) \\
\text{else } \text{filter}(\text{tail}(xs), test)
\end{cases}
\]
Systems Supporting First-Class Functions (Selection)

**Programming languages**

Functional programming languages: Lisp/Scheme, Haskell, ML/OCaml/F#

Scripting languages: JavaScript, Python, Ruby, Perl

**Composition systems**

Common Music, OpenMusic, SuperCollider
Recapitulate the following concepts, for example, with these slides, programming textbooks (e.g., from your former programming classes), or even the internet.

- Data abstraction
- Object-oriented programming
- Higher-order programming

Recommended literature for studying these concepts (and programming in general) more deeply: Abelson and Sussman (1996). Structure and Interpretation of Computer Programs. MIT press.

Music representation is a complex task

- Various information to express, which is influenced by several factors (e.g., musical style, requested music theory concepts)
- Computer science concepts help to organise representation: e.g., hierarchic representation, data abstraction, object-oriented programming, higher-order programming