Chris Hogan

Wesley Hoffman

EECS 647 Project Report 1

10/2/2014

1. Scenario and Mini-World Description

Our project is called RandoMIDI. It is a web application that generates pseudo-random MIDI files based on user specified parameters. For example, a typical use case would look something like this:

> A user visits the home page and clicks a button to begin the composition process. A form is presented with various options such as instruments, tempo, style, length, key, time signature, mode, etc. Say the user decides to choose 'classical' style with 'piano' and 'cello' for instruments in the key of C major. Any form elements left blank will be selected at random. The application will take this information and use it to construct a MIDI file using random elements specified in the database, but keeping to the parameters. The result is a composition that can be played in the browser.

The more interesting use case would be to see what kind of weird pieces get generated when leaving everything random, or choosing conflicting elements. We see this as an inspirational tool for musicians and composers, as well as a learning tool for anyone who loves music.

Details of Composition Creation

In the example above, the application would first identify the style. The database will have different motifs, rhythms, progressions, instruments, and harmonies associated with each available style. In this case, it will assemble a random chord progression in C major that roughly fits with a classical style, choose motifs that harmonize that progression, use predefined, simple rhythms repeatedly, and generate a MIDI file with the help of a MIDI processing library.

Mini-World Description

The RandoMIDI project has many elements. These elements are Motifs, Notes, Instruments, Songs, Chords, Progressions, Rhythms, RhythmDuration, Scales, Durations, and InstrumentsInSongs. Motifs contain and identifier, a name, octave range, tone, and an accidental. A motif has notes. Notes contain octaves, tones, and accidentals associated with them. Instruments contain a name, range, and role. The range of the instrument consists of notes. Songs contain a name and a tempo. Songs have notes, scales, chords, progressions, and instruments in the song. Chords contain a name. A chord has notes and a duration. It can also be a part of a scale. Progressions contain an identifier and a name. Progressions have a number of chords. Rhythms contain an identifier, measures, and time signatures. Rhythm Durations contain an identifier and a name. Scales contain a name and a mode. They also have notes and the range differs based on the instrument. Durations contain a rest length and a name. InstrumentsInSongs contains a name.

2. Requirements Analysis – Data and Operations

- **Motifs**
  A motif is the smallest musical element. It is a short pattern of rhythmic and/or tonal elements.

  Operations
  - select_random_motif(): Chooses and returns a random motif from the motif table.
  - create_motif(): Allows developers (maybe users) to add a motif to the database.
  - select_motif(style): Returns a motif that matches the style argument.
  - generate_random_motif(): Creates a random motif and adds it to the database.

- **Notes**
  This is a collection of all possible musical notes within the range of the standard, 88 key piano. This includes all accidentals.

  Operations
  - add_to_motif(note): Adds the argument note to the set of notes in an existing motif.
  - remove_from_motif(note): Removes the specified note from an existing motif.
  - get_notes_for_scale(scale): Returns a list of all notes in the specified scale.
  - up_octave(note): Returns the given note, raised one octave.
  - down_octive(note): Returns the given note, lowered one octave.
  - stepwise_note(note): Returns a note for a stepwise sequence.

- **Tempo**
  This is a collection of all available tempos, along with ranges for the classical names (Allegro, etc.). We will allow 40 bpm to 240 bpm.

  Operations
  - increase_tempo(bpm): Increments the current tempo by bpm.
  - decrease_temp(bpm): Decrements the current tempo by bpm.
  - set_tempo(bpm): Sets the tempo to bpm.

- **Instruments**
  Collection of all instruments we'll allow, along with their range, and any associated MIDI information.

  Operations
  - add_instrument(inst): Adds the specified instrument to a composition
  - remove_intrument(inst): Removes the specified instrument to a composition.

  - chorus(instr): Creates a chorus effect for the specified instrument.
  - solo(instr): Creates a solo instrument sound.

- **Rhythms**
  Holds rhythmic patterns that can be used for motifs or styles. These will typically be just a couple measures long or shorter. Durations will be specified by w (whole note), h (half), q (quarter), e (eighth), s (sixteenth).

  Operations
  - get_random_rhythm(measures, time_sig): Returns a random rhythm that is of length *measures* and fits the specified time signature.
  - add_rhythm(rhythm): Adds specified rhythm to the database.

- **Scales**
  Holds the notes and steps associated with the common scales in western music. Notes and step values (whole or half) need to be stored.

  Operations
  - create_scale(scale): Adds user generated scale to the database.
  - generate_motif(scale): Generates a pseudo random motif that fits the scale.

- **Progressions**
  Stores standard classical chord progressions for major and minor modes. Progressions will be represented as a finite state machine: a predominant moves to a dominant, a dominant moves to a tonic or third, etc.

  Operations
  - move(state): returns a list of available chords to move to from the current chord (*state).*

- **Chords**
  Represents the possible combinations of notes to form the most common chords: major and minor triads, seventh chords, etc.

  Operations
  - augment(chord): Returns an augmented version of the specified chord.
  - diminish(chord): Returns a diminished version of the specified chord.
  - harmonization(chord): Returns possible melody notes that harmonize the chord.

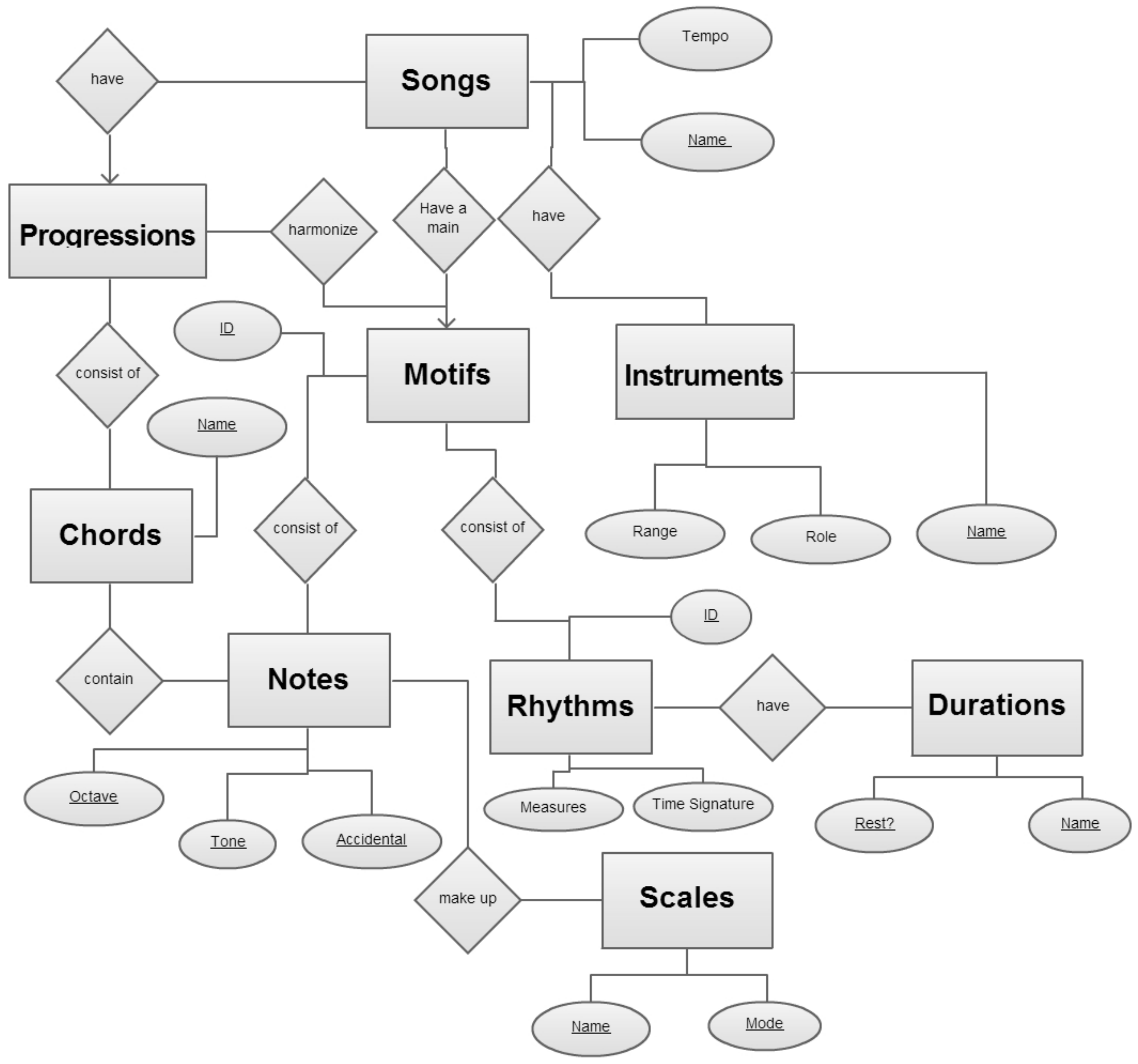
- **Styles**
  Representations of different styles of music. This includes typical rhythmic patterns, tempos, scales, chords, instruments, etc. For example, if a user wants a MIDI file in a "blues" style, this would mean guitar drums and base, 12-bar blues organization, I-IV-V progressions, and blues scales.

Operations
- add_style(style): Allows for adding user generated styles to the database.

- **Time Signatures**
Stores all available time signatures.

Operations
- double_time(time_sig): Changes the current time signature to double time.
- half_time(time_sig): Changes the current time signature to half time.

3. ER Diagram



Constraints

One song has one progressions. A chord has two to five notes. A scale has five to seven notes. Motifs should only have a few notes and rhythms.

4. Relational Schemas

Songs(Name: string, Tempo: int)

Chords(Name: string)

Progressions(ID: int, Name: string, Name: string)

Notes(Octave: int, Tone: char, Accidental: string)

Motifs(ID: int, Name: string, ID: int, Rest?: Boolean, Name: string,

Octave: int, Tone: char, Accidental: string)

Rhythms(ID: int, Measures: int, Time_Signatures: int)

RhythmDuration(ID: int, Rest?: Boolean, Name: string)

Scales(Name: string, Mode: string)

Durations(Rest?: Boolean, Name: string)

Instruments(Name: string, Range: string, Role: string)

InstrumentsInSongs(Name: string, Name: string)

Many of these schemas don't fit well into the ER model. For example, a progression can be an arbitrary sequence of chords, which can be made up of an arbitrary group of notes. Here, we are generating data on the fly instead of trying to keep referential integrity. However, once a chord progression has been identified, we would like to save it in the database with a unique ID. For example, the progression I-V-I in the key of C major would have the chords C G C, C consisting of the notes C, E, G, and G consisting of the notes G, B, D. Since a progression can be anywhere from 2 to an arbitrarily long number of chords and each chord can be 2 to 13 notes, it doesn't make senses to try use the ER model for this. Many of our relations have this property. Another example is Motifs. Each song has a main motif, which consists of a few notes in a specified sequence, each with its own rhythmic value. We're unsure how to represent this in the ER style. Perhaps most of our data will be in memory as we operate on it to produce our MIDI files. It's difficult to say at this point.