HAIL – Domain Model

Sebastian Fichtner

1 Domain Objects

PlayInfo

- volume : float
- pan : float
- delaySeconds : float
- offsetSeconds : float

mixing and timing info for a play instruction

PlayableProtocol

- play(playInfo : PlayInfo) : void
- pause() : void
- stop() : void
- lengthSeconds : float

SchedulingAudioFile <PlayableProtocol>

- location : URL

This audio file enqueues play instructions for future playing. play(...) can be invoked multiple times, even when the first instruction hasn't been played yet. The instructions must not overlap and must be enqueued in chronological order.

Music <PlayableProtocol>

- name : String
- baseKey : int
- bpm : float
- playInfo : PlayInfo

Has its own play info that it applies in addition to the play info it recieves. Most of the time, delay and offset will be 0.

MusicReference : Music

- music : Music*

Sample : Music

- audioFile : SchedulingAudioFile

Instrument

- name : String
- keyMusicReferences : MusicReference[]

InstrumentReference : Instrument

- instrument : Instrument*

ScoreEvent

- startTimeSeconds : float
- durationSeconds : float
- key : int

Score

- name : String
- lengthSeconds : float
- bpm : float
- baseKey : int
- events : ScoreEvent[]

ScoreReference : Score

- score : Score*

score reference inherits from score. it has its own name, length, bpm, baseKey and events. that means it transforms the referenced score to its own values. in case of the eventlist it works like that: for every event of its own, it plays the whole referenced score at the key of that event.

the score reference can have invalid values for its parameters. in that case the own parameter is not applied and the parameter of the referenced score is used unaltered. if all parameters are invalid (default), the score reference is just an encapsulated pointer to a score.

Performance : Music

- instrument : InstrumentReference
- score : SoreReference

GroupPerformance : Music

- performances : Performance*[]

1.1 Note on References

References serve two purposes:

- Indirection: in order to built his music in a modular orthogonal nonredundant manner, the user needs to have a pointer type for the objects he uses. he uses it to bundle pointers that are supposed to point to the same object. when he wants to change or exchange that object he only has to do that once.
- Transformation: We also need the logic of that bundleing for mixing, and we want to be able to apply more transformations than just volume and pan adjustments.