Hear the Garbage Collector: a Software Synthesizer in Java "Harmonicon"

An overview of the project implemented for IBM Research

Guest Speaker:
Florian Bömers
Founder, bome.com

Universität Salzburg
Fachbereich
Computerwissenschaften

Speaker Introduction

- Florian Bömers owns a software company specializing in MIDI and audio tools
- Research and development of digital audio software since 1986; in 2000 diploma thesis about real time audio processing with wavelets
- From 2001-2004, he was leading Java Sound development at Sun Microsystems
- Active in a number of open source projects about Java Sound and audio software
- He works as consultant for various companies in the field of audio/media software architecture

Agenda

- Introduction
- Goals
- Technologies
- Synthesizer Architecture
- Realtime Java integration
- Results so far
- Demo
- Outlook
- Q&A/Discussion

Introduction

The idea:

- Garbage Collectors interrupt the VM
- for audio playback, interruptions cause
  - bad timing
  - audible gaps
- a real time software synthesizer well suited:
  - should allocate a lot of objects
  - will expose intrusiveness of garbage collectors

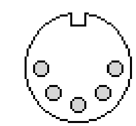
➔ hire Florian to implement such a software synth

Goals

Before the implementation was started, these goals were fixed:

- implement a full real time synthesis engine in pure Java
- adhere to standards
- optimize for high end, i.e. enable very high quality (192KHz sampling rate, 64-bit float samples, 5.1 channels, no polyphony limitation)
- achieve 1 millisecond latency (possibly with custom sound driver)

Technologies: MIDI

- **M**usical **D**evices **D**igital **I**nterface
- Realtime protocol to control electronic music instruments
- Industry standard since 1982
- Standardized MIDI cable with 5-pin DIN plug 
- Low bandwidth (31250bits/sec)
- Send semantic commands rather than abstract sound
- Sound generator required to hear the MIDI commands
- Data is not queued/scheduled -> realtime
- High requirements regarding jitter and latency

Technologies: MIDI cont.

- Different classes of commands
- Channel commands: Note On, Note Off, change instrument, ...
- Realtime commands: Reset, ...
- Channel commands are assigned to one of 16 logical channels
- Multi-timbral synthesizers play commands on different channels with different instruments simultaneously

Technologies: MIDI Files

- SMF – Standard MIDI File
- Standardized 1991
- Store MIDI commands in a file along with timing information
- Very efficient storage, ~10KB per minute
- E.g.: accurately capture a live keyboard performance
- Extensive editing possible, print the music, etc.
- But: audio quality depends on tone generator!

Technologies: General MIDI

- Standardized set of 128 instruments
- Enables exchange of MIDI files – all General MIDI tone generators will play the file with the right instruments
- Still: audio quality depends on tone generator!
- Was extended to General MIDI 2, with several banks of 128 instruments each

Technologies: SoundFont I

- Industry Standard by Creative Labs
- MIDI to WAVE: take structured MIDI commands and create a stream of (abstract) digital audio
- Synthesis standard
- Usually implemented in hardware on Creative Labs soundcards
- SoundFont files are exchanged on the Internet
  - quality vs. size
  - number of instruments
  - unusual instruments

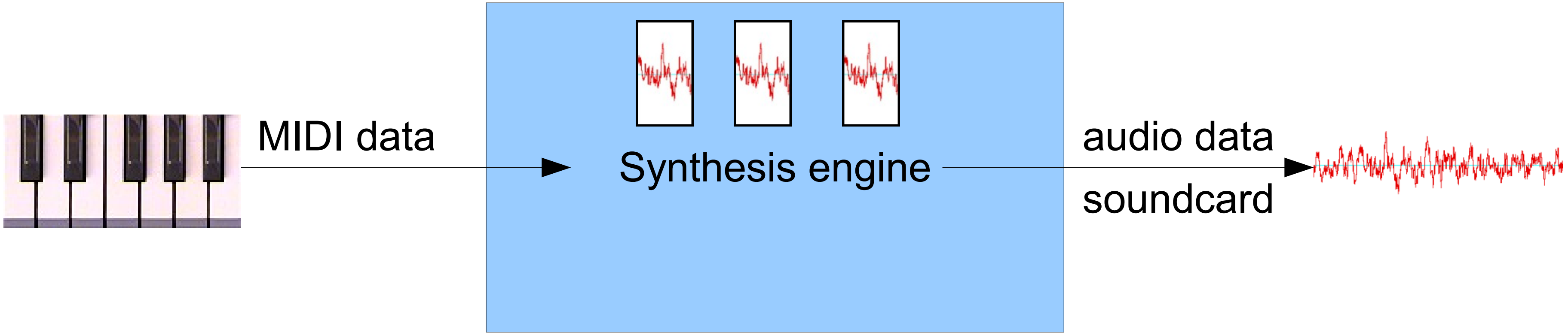
Technologies: SoundFont II

Based on wave table:

- Each instrument is stored in form of a short recording (wave file)
- When a note is triggered, the wave file is played
- Higher pitch is achieved by playing the file a little faster, lower pitch vice versa
- Powerful meta data allows far reaching processing of the stored wave files – loop portions, change volume curve, apply frequency filters, change pitch curve

Architecture: Overview

- The synthesizer continuously renders small chunks of audio data
- Each chunk contains the audio data for the MIDI notes
- The small chunks are written to the soundcard
- The size of the chunks determines the latency



Architecture: MIDI input I

- When a MIDI command enters the synthesizer, it is time-stamped with the current real time
- It is added to a queue of MIDI commands
- From the synthesizer thread (in regular intervals) the queue is read and processed:
  - for an instrument change, change the corresponding value in the MidiChannel object
  - for a Note On, insert a new note (next slide)
  - for a Note Off, release the existing note (next slides)

Architecture: MIDI input II

Note On:

- A Note object is created:
  - from the SoundFont wave table, find the wave of this note's instrument
  - from the SoundFont instrument definitions, retrieve the meta data for this note, i.e. volume/pitch curves, fine tune, etc.
- Insert the Note object as input stream into the AudioMixer

Architecture: MIDI input III

Note Off:

- In theory: find the corresponding Note object and remove it from the mixer
- But: would cause very abrupt ending of the note, possibly with click
- Rather, the Note enters the *release* phase: defined by the SoundFont meta data, usually a fade out

Architecture: Audio Mixer I

- The AudioMixer is a “naive” class that owns a list of AudioInput objects
- When the AudioMixer is asked to mix a buffer of audio data, it mixes this buffer worth of audio data from all AudioInput objects
- ```
  public interface AudioInput {
     public void read(AudioBuffer buffer);
  }
  ```


  ```

Architecture: Audio Mixer II

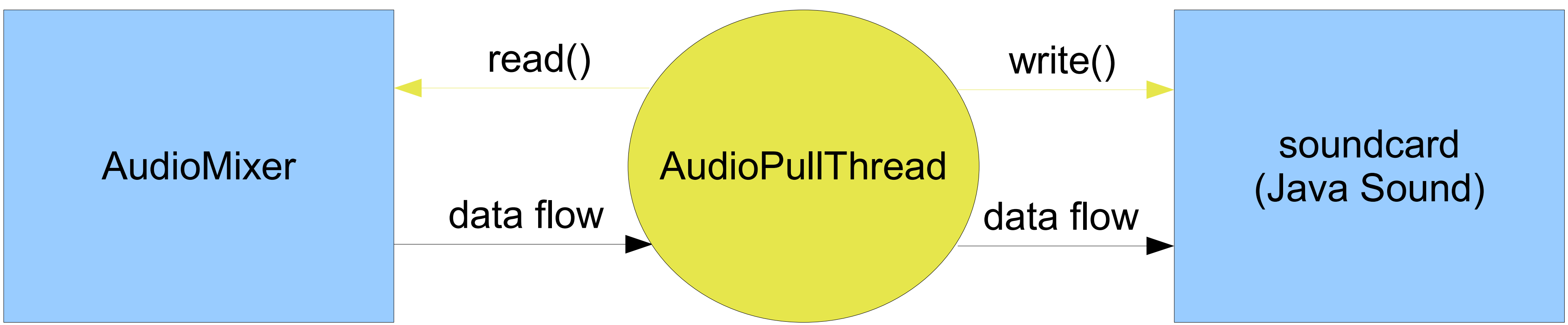
- AudioBuffer is a wrapper for a double[] array
- AudioMixer implements AudioInput, can cascade AudioMixers
- “Pull” architecture: the AudioMixer pulls from the AudioInput objects

Architecture: Audio Mixer III

- The Note class implements AudioInput: to the mixer, they are just an object that provides an audio stream
- This abstraction allows heterogeneous synthesizers, e.g. with different synthesis engines
- The number of current AudioInput objects in the mixer is the current polyphony

Architecture: Soundcard

- The master thread continuously
  - reads an audio chunk from the mixer (“pull”)
  - writes the chunk to the soundcard (“push”)



Architecture: Queuing?

- The synthesizer needs to queue 1 audio buffer worth of MIDI commands
- E.g. if the audio buffer size is 100 milliseconds:
  - every 100 milliseconds, a buffer with 100 milliseconds of audio is rendered
  - already at the beginning of a 100ms period, the engine needs to know the sound of the end of this period
  - this is e.g. because each Note renders a full 100ms buffer at once
  - synth always lacks 100ms behind real time
  - needs to queue MIDI commands for 100ms

Architecture: Synchronization

- Since MIDI data is buffered, need to make sure that all MIDI commands are processed in order
- Need to make sure that a Note On followed by a Note Off in the same buffer will still create the Note and make it audible
- Need a stable clock: usually use the soundcard's clock
- If the system clock is used to time-stamp MIDI data, need to synchronize soundcard time with real time (compensate drift)

Architecture: Performance I

- In the example of 100ms buffers:
  - 100 simultaneous notes require that each Note object renders 100ms worth of audio data in just 1ms
- But...100ms is much too long:
  - project requirement: 1ms buffers
  - trained (human) ears can distinguish rhythmic errors as small as 1ms
  - latency of more than 10ms is disturbing for live keyboard performance

Architecture: Performance II

- The smaller the buffers, the more overhead:
  - loop initialization, jumps
  - MIDI data processing always at buffer boundaries
  - status checks
  - more calls to write the data to the soundcard
- the smaller the buffers, the more fragile the synthesizer becomes, and the more it will require real time scheduling
- If rendering comes late or takes too much time, the soundcard will receive the next audio buffer too late (buffer underrun), causing a short pause (click)

Architecture: Performance III

Parallelization:

- Rendering each Note object is independent of other notes
  - perfect for parallel execution on multiple processor cores
- The synthesizer implements a scheduling algorithm for multi-threaded rendering

Architecture: Performance IV

Soundcard driver:

- Java Sound's audio output is general purpose, not suited for very low latency
- Windows' Direct Sound has minimum latency of 23ms
- On Linux, Java Sound uses ALSA, 5ms possible
- Needed a custom driver to directly talk to the soundcard:
  - Linux implementation only, currently
  - use ALSA directly
  - low overhead

Real Time Java Integration I

- Use IBM's Eventrons:
  - high frequency thread with hard scheduling
  - suited to drive the main rendering thread
  - even better suited to write rendered audio data to the soundcard, “sample by sample”
- However, adds more synchronization points

Real Time Java Integration II

- Garbage collector (GC) problems:
  - if GC interrupts right before a MIDI event is time-stamped, the time stamp will be off
  - GC may interrupt enough to cause the rendering thread to take too much time
  - GC may interrupt at the moment where the rendered buffer is about to be written to the soundcard, so it will come too late
- Therefore, Harmonicon will greatly benefit from the Metronome GC

TuningFork Integration

- Integrate Harmonicon into TuningFork, IBM's visualizer of garbage collector trace files
- played MIDI commands are available as staff view
- it can be easily seen if musical delays originate from the garbage collector
- See garbage collector activity and notes during playback

Status: Features

- Almost full SoundFont 2.01 standard implemented (missing: some interactive controls, chorus and reverb effects)
- Can play back real time MIDI and MIDI files
- GUI with some interactivity
- Eventron support
- Basic TuningFork integration

Status: Performance

- Rendering benchmarks on AMD 4400+, dual core, stock IBM VM on Windows:
  - normal MIDI file in 40x realtime
  - up to 850 note polyphony
- With direct ALSA audio driver:
  - stereo rendering, 10 channels written to soundcard
  - 192KHz sampling rate
  - 8 samples buffer size -> theoretical 40ɥs latency (higher in practice)

ALSA: Advanced Linux Sound Architecture (Linux audio driver model)

Status: Hearing the GC?

- Harmonicon does not allocate a lot of objects during normal operation:
  - every MIDI command is one object
  - Note object are instantiated individually
  - some other smaller objects
- Needed to add some allocations in order to let the GC kick in
- Then, stock garbage collectors were quite disrupting
  - yes, we can hear the GC!

Status: Real Time Java

- Eventron support works, but with current alpha version of Eventrons does not increase performance
  - Harmonicon will be useful for testing and optimizing eventrons
- Running Harmonicon on the Metronome VM was not possible yet, since no Metronome VM with JIT was available

Status: Harmonicon in Concert!

- Perry on keyboard attached to a computer running Harmonicon
- Florian on cello
- David supervising the computer

Demo

- Listen to Harmonicon

Outlook

Some things still to be done:

- physically measure exact latency (rather than believing the computer)
- multi channel/surround support
- implement MIDI effects
- optimize MIDI input with own driver
- full TuningFork integration

Discussion

Florian.Bomers@bome.com