Distributed Music-Jam with Java™ Sound and Project JXTA

Florian Bomers

Java Sound Technical Lead
Sun Microsystems, Inc.

Primary Purpose of this BOF

Get to know the jam application.
Learn some tricks for programming Java Sound (and Project JXTA)

BEGINNING

Speaker

- Florian Bomers leads development of Java Sound at Sun Microsystems
- Responsible for MIDI API and implementation of the MMAPI (J2ME)
- Profound Java experience for 8 years
- Key contributor to jsresources.org and Tritonus

BEGINNING

Key Topic Areas

- Demo: Jam
- Java Sound issues
- ogg/vorbis issues
- Project JXTA programming *)

Unfortunately no material for that

BEGINNING

Demo

AudioFormat 1

- Jam needs concurrent playback and recording
- Some computers only allow to record in specific audio formats, possibly different from playback format
- Solution: all audio streams are converted to one common playback format upon loading/recording
- Needs Tritonus' SamplerateConverter and PCM2PCM converter for flexible conversion

MIDDLE

AudioFormat 2

- Semi-smart algorithm:

1) If non PCM, convert to PCM,16-bit
   For ulaw, alaw, mp3, and ogg/vorbis
2) If bit size/signedness does not match playback format, convert sample size and signed-ness with PCM2PCMConverter
3) If sample rate doesn't equal playback samplerate, convert sample rate by using SamplerateConverter

MIDDLE

Metronome

- MIDI would provide an easy method to output the metronome sound
- Difficult to synchronize MIDI with audio
- Solution: Metronome is handled as an internal track
- The metronome audio data is created in real time from a “tock” wav file

Mixing 1

- Java Sound provides a software mixer.
- Could use one line per track and let Java Sound mix
- Problems:
  - Synchronization of different lines not implemented in Java Sound
  - Some Java Sound implementations may not provide software mixing

Mixing 2

- Solution: implement own mixer
- Use FloatSampleBuffer to hold single tracks
- FloatSampleBuffer a utility class from Tritonus to convert and handle audio in float point samples
- In a loop, mix the same portion from all tracks to the mix buffer (also a FloatSampleBuffer)
- Then write the mix buffer to the audio device
- -> perfect synchronization

MIDDLE

Synchronization: Play Position 1

- Problem: at playback, how can we get the exact playback position?
- DataLine has methods to query position
- But returned position is very inaccurate
- Solution: employ a mixture of guessing and interpolation...

Synchronization: Play Position 2

- known: the number of samples written to the device
- At each SourceDataLine.write(), measure the current time: posCurrTime
- Playback position =
  written samples in millis
  + ((current time) - posCurrTime)
  - lineBufferSizeInMillis
  - hardcodedDelay

MIDDLE

Synchronization: Play Position 3

- Problem: how to get the current time in millis?
- Bad idea: System.currentTimeMillis()
  - on Windows only 50-60millis resolution
- New! With Java 1.4.2, use sun.misc.Perf
  - high resolution timer
- But beware: security protected, not a public API

Synchronization: Play Position 4

- Solution: wrapper method that uses either the new one or the old one:

```
private static boolean use142 = true;
private static sun.misc.Perf perf = null;
private static long freq;

public static long getCurrentTime() {
    if (use142) {
      try {
        if (perf == null) {
          perf = sun.misc.Perf.getPerf();
          freq = perf.highResFrequency();
        }
        return perf.highResCounter() * 1000 / freq;
      } catch (Exception e) {}
    }
    use142 = false;
    return System.currentTimeMillis();
  }
```

Synchronization: Record<->Play

- Problem: when recording a new track, how to synchronize the recorded data?
- Solution: start the TargetDataLine before actual recording is supposed to start.
- Then discard all samples until actual recording is supposed to start
- Remaining problem: when exactly stop discarding samples?

MIDDLE

ogg/vorbis decoder 1

- Silence is encoded very efficiently. Small amount of ogg data may expand to huge amount of PCM data.
- Decoding an ogg page at once can flood the internal circular buffer and cause a deadlock.

MIDDLE

ogg/vorbis decoder 2

- Solution:
- preliminary:
  Increase size of circular buffer by factor of 10.
- Better solution (to be implemented):
  Handle data in ogg packets (ogg packets are smaller than ogg pages), fill buffer only up to a certain degree.

MIDDLE

ogg/vorbis encoder 1

- Problem: vorbis paradigm of requesting buffer from native library/acknowledging used amount doesn't fit well into Java's handling of arrays
- vorbis returns pointer to native 2-dimensional float array:

```
float** buffer
     = vorbis_analysis_buffer(&dsp_state, 1024);

// write to buffer[channel][sample 0..979]

vorbis_analysis_wrote(&dsp_state, 980);
```

MIDDLE

ogg/vorbis encoder 2

- Solution: Always copy data, Java layer creates 2-dimensional buffer and passes it to the native layer
- Java code:

```
float[][] buffer = new float[channels][1024];
// write to buffer[channel][sample 0..979]
dspState.write(buffer, 980);
```

MIDDLE

Where's Project JXTA?

- Problem: Matthias became sick
- Stayed the last week in hospital
- Solution: next Java One ?
- Or:

Project JXTA Town Hall Meeting on Wednesday

June 11, 6:00pm - 8:00pm PST

Renaissance Parc 55 Hotel in San Francisco, CA

The evening will start out with Rob Gingell, Sun Fellow and Vice President, addressing Java communities which include Project JXTA, Jini, the Java Community Process program. Then we'll go into our own meeting room and talk about Project JXTA. Food and drinks will be provided.

Q&A

JavaOne℠

Sun's 2003 Worldwide Java Developer Conference℠

JAVA™

java.sun.com/javaone/sf