The Java Sound Internet Phone

java.sun.com/javaone/sf

Florian Bomers
bome.com

Overall Presentation Goal

Media Player with Rewind and Record buttons

Learn how to build a simple Media Player application with Java.

Explore how to leverage Java Sound in J2SE 5.0.

Speaker Introduction

- Florian Bomers owns a company specializing in MIDI and audio tools.
- Until last year, he was leading Java Sound development at Sun Microsystems.
- He has been programming with the Java Sound API since its very beginning.
- He is co-leading the Tritonus project – an open source implementation of the Java Sound API, and plugins.
- He co-founded the jsresources.org project (Java Sound Resources): open source FAQs, examples, applications.

Agenda

Demo

General Architecture

Program Details

Problems and Solutions

Future Enhancements

Your Questions

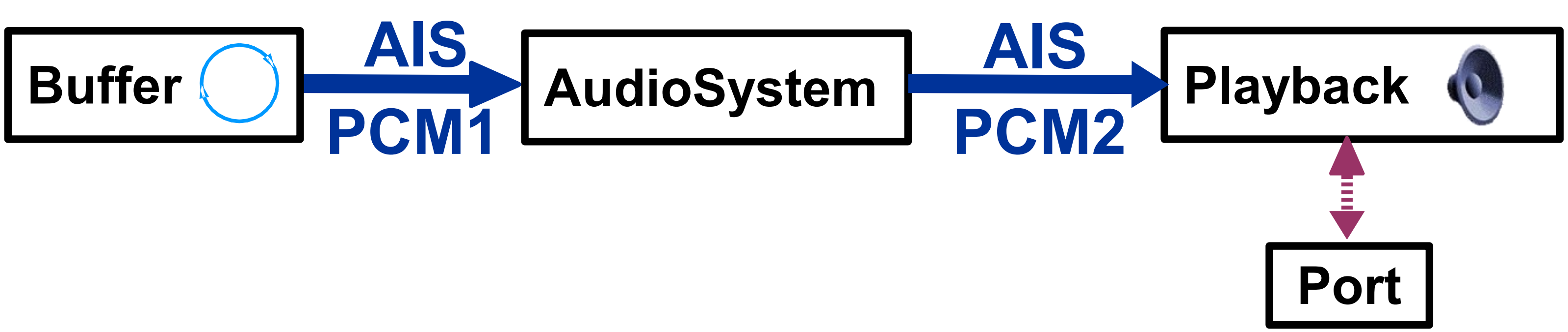
Demo

The Time Turner in Action!

General Architecture

Sound Output I: Playback

- Uses Java Sound's `AudioInputStream` architecture to stream audio data
- Use Java Sound's SPI mechanism to use ogg, mp3, and gsm plugins
- Use `SourceDataLine` in package `javax.sound.sampled` for actual streaming playback on audio device



General Architecture

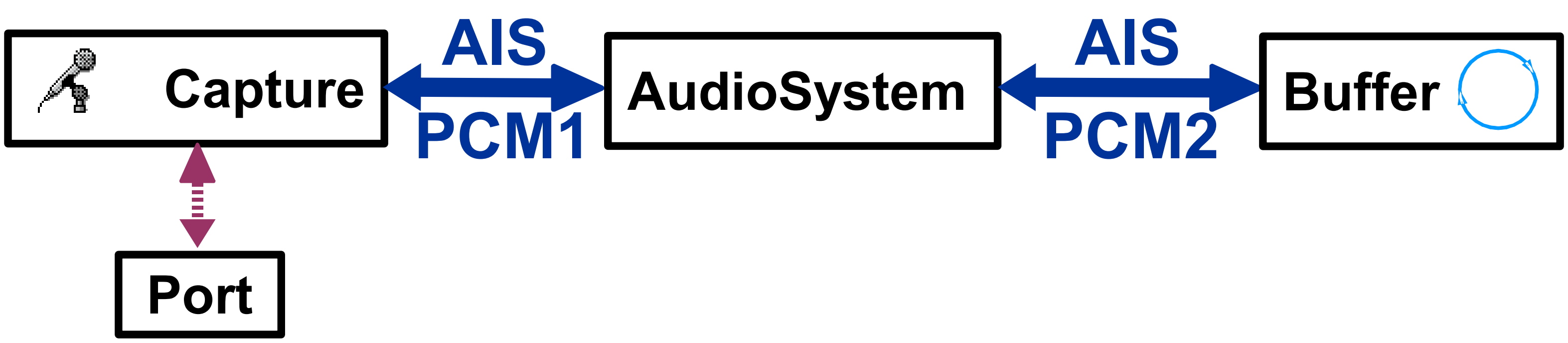
Sound Output II: Recording

- Uses Java Sound's `AudioInputStream` architecture to stream audio data
- Use Java Sound's SPI mechanism to encode to ogg, mp3, and gsm formats
- Use `AudioSystem` to write audio data to file

General Architecture

Sound Input 1: LINE IN

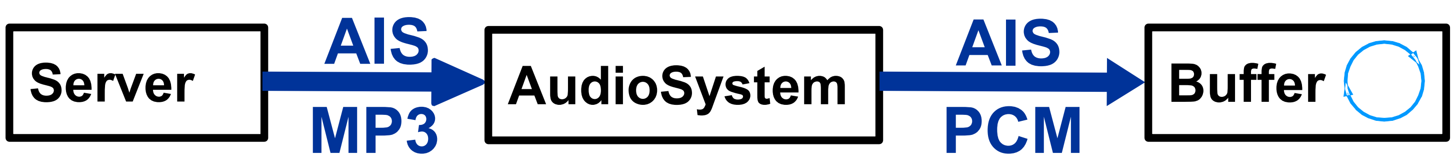
- connect a regular radio device to the LINE IN port of the computer
- Use Java Sound's `TargetDataLine` to capture live audio data
- Use `Port` to select the appropriate plug on the computer



General Architecture

Sound Input 2: Internet Radio / Shoutcast

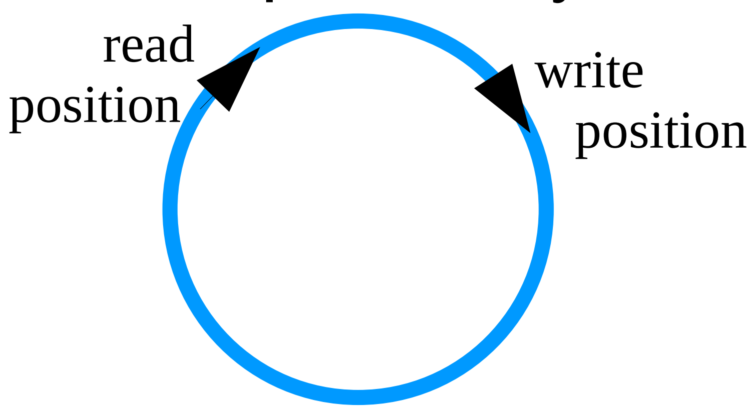
- use a TCP connection to connect to shoutcast/icecast server
- wrap the stream in an `AudioInputStream` and use Java Sound to decode the mp3 or ogg stream



General Architecture

Turning Time I: the Circular Buffer

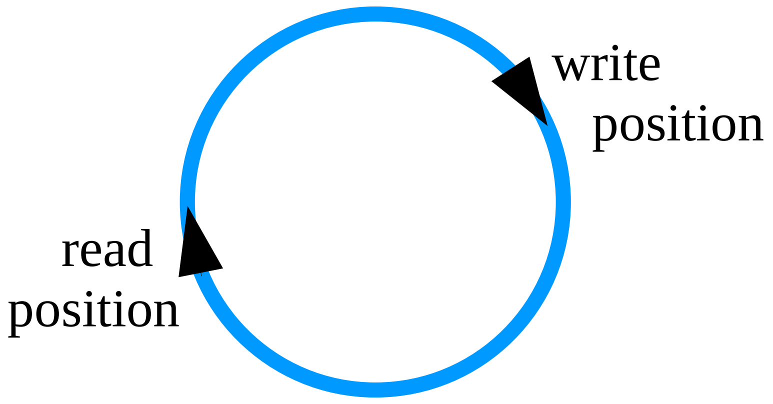
- use a special circular buffer to intermediately store all audio data that's coming in
- size of buffer is configurable, e.g. 30 seconds
- once the buffer is full, new incoming data overwrites the oldest (e.g. 30 seconds old) data
- Reading occurs independently from writing to it



General Architecture

Turning Time II: the Circular Buffer

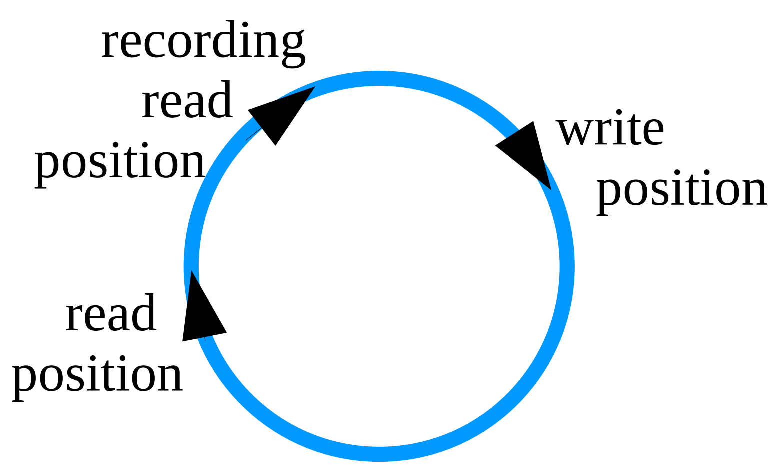
- for rewinding, move the read position back
- since it reads at the same pace as data is fed to the circular buffer, the 2 positions keep their same distance
- forward works the opposite way



General Architecture

Turning Time III: the Circular Buffer

- a separate, independent read position for recording
- cannot use the read position for recording, because winding should still be allowed during recording



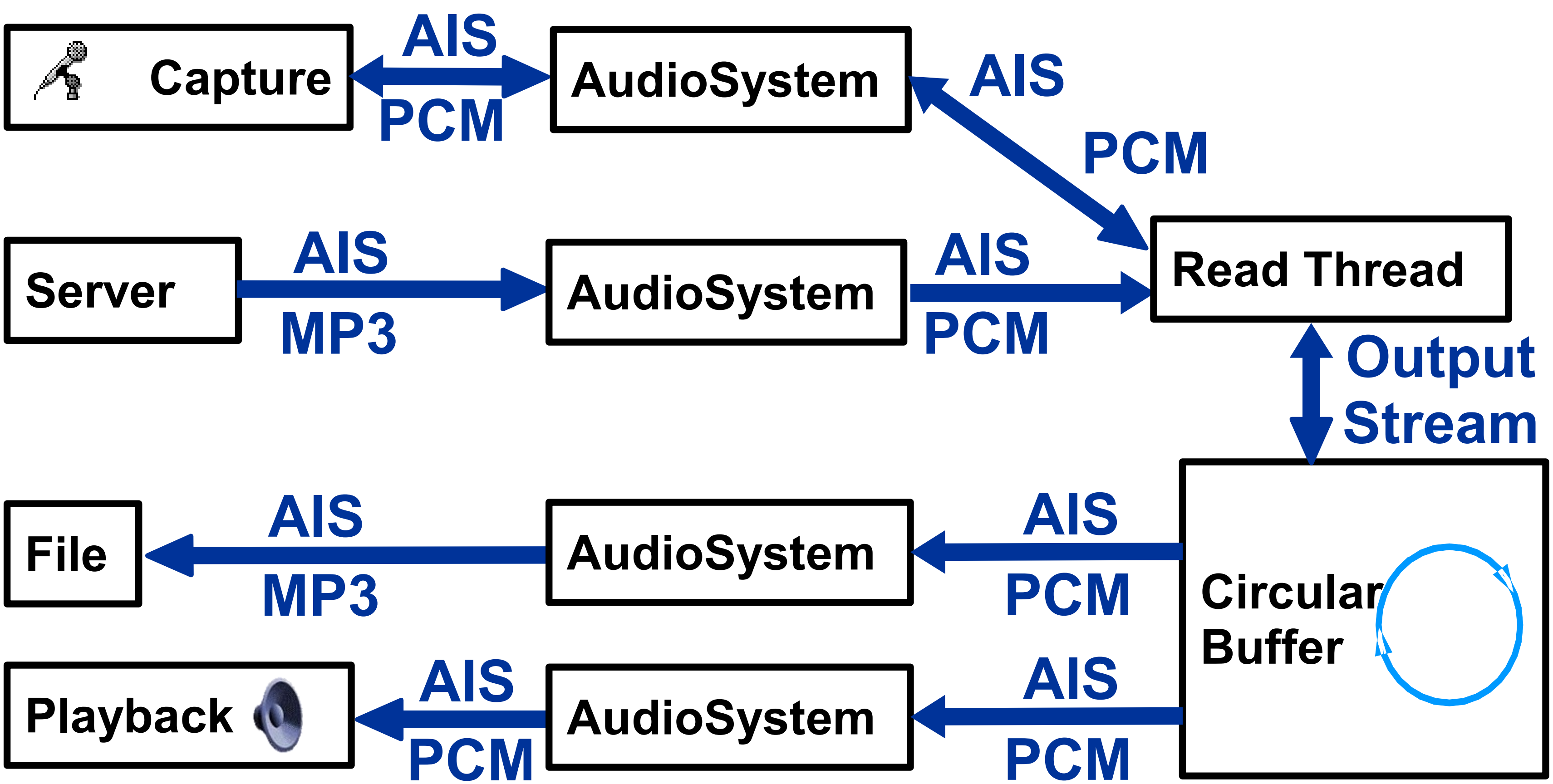
General Architecture

Audio Flow Overview

- use `AudioInputStream` for audio flow
- input: wrap network data or `TargetDataLine` into an `AudioInputStream` for conversion to PCM format
- circular buffer is an `OutputStream`, the incoming audio is written to it at the write position (in an own thread).
- circular buffer provides a special `AIS` for reading from the read position
- recording gets a second `AIS` from the circular buffer for reading from the recording position

General Architecture

Audio Flow Diagram



Program Details

Different Audio Formats

- several audio formats involved:
  - network, e.g. mp3 from shoutcast
  - capture format: PCM
  - circular buffer format, fixed to PCM
  - speaker line format, PCM (maybe different from circular buffer)
  - recording file format, e.g. ogg
- use `AudioSystem.getAudioInputStream` to convert the different formats to and from the circular buffer format

Program Details

Winding

- Since reading from circular buffer is implemented with special instances of `AudioInputStream`, its `skip()` method is used for winding:

```
public void wind(int millis) {
    int bytesToSkip = AudioUtils.millis2bytes
                          (millis, ais.getFormat())
    try {
        ais.skip(bytesToSkip);
    } catch (IOException ioe) {
        // nothing to do
    }
}
```

Program Details

Mixers, Ports I

- User can choose the `Mixer` for capture and for playback
- User can choose the `Port` to capture from:
  - a list of all Ports
  - a volume slider
  - will select it as capture source
- User can change the playback volume of an arbitrary `Port`:
  - a list of all Ports
  - a volume slider
  - does not select which output port is used for playback!

Program Details

Mixers, Ports II

- Partial implementation of Ports available in Java since J2SE 1.4.2
- Since JDK 1.5: available on all platforms
- You can select the input port on soundcard
- Adjust volume for input and output ports
  - gain, pan, mute, select

Program Details

Direct Audio

- use Direct Audio devices (J2SE 5.0) for high performance
- “direct” access to the soundcard
  - access to all soundcards
  - low latency (small buffers) possible
- implementation for Linux available in J2SDK 1.4.2
- Since JDK 1.5: available on all platforms
- Requires ALSA on Linux, DirectSound 5 on Windows, and *mixer* on Solaris

Program Details

Graphical Level Meter

- misuse a `JProgressBar` as level meter
- do a simple "max" analysis on every audio data block that passes the system (`DataLine.getLevel()` is not usable!)
- a dedicated thread: every 50 milliseconds it reads the current level and displays it as "progress"
- for nicer display, limit the amount it can decrease

Problems and Solutions

Port assignments

- Problem: can access all `Port`s on the system, but which `Port` really selects, e.g., LINE IN?
- Which port will adjust the volume for the selected `Mixer` and `SourceDataLine`?
- ➔ Java Sound does not provide information which `Port` instance controls which `DataLine`.
- ➔ In our application, we present the user a list of all `Port`s, and she/he needs to pick the correct one to be able to adjust the gain/volume.

Problems and Solutions

Changing Buffer Size while line is running

- In order to find out the minimum buffer size (i.e. minimize latency), we want to change the buffer size while audio is playing
- But buffer size is specified when opening the Line!
- ➔ Close the Line, and re-open it with the new buffer size.
- ➔ Not nice, but unavoidable. API should be enhanced to allow buffer changes while Line is open.
- ➔ Analogous for changing the current `Mixer`.

Problems and Solutions

Circular Buffer

- after recording started, playback position may be changed arbitrarily, but recording position needs to advance continously
  - 2 circular buffers with same data?
  - 2 read pointers?
- would like to use Tritonus' circular buffer implementation
- but high overhead and complexity with 2 circular buffer instances

➔ so a custom circular buffer was implemented with 2 read pointers

Problems and Solutions

Circular Buffer

- if we keep the read pointers as an absolute position in the circular buffer, it may get overwritten when more data is written to it than it is read
- ➔ keep read pointers as relative position ("distance", lag) to write position
- ➔ writing to circular buffer must increase the distance to the read pointers
- ➔ reading from buffer decreases the distance

Problems and Solutions

Re-encoding?

- incoming shoutcast stream is encoded (mp3 or ogg)
- stream is always decoded to PCM for the circular buffer
- writing to file will re-encode it
- lower audio quality, high processor usage, high memory usage
- → design decision to implement the circular buffer as a PCM-only container for simplicity
- → app can be optimized in future to writing the original stream "as is" to file

Problems and Solutions

Writing to File I

- `AudioSystem.write()` blocks until all data is written to file
- ➔ use an own thread that calls `AudioSystem.write()`
- ➔ `AudioSystem.write()` blocks for the entire duration of writing to file
- ➔ When the `AudioInputStream` returns -1 in its `read()` method, writing to file is finished

Problems and Solutions

Writing to File II

- Writing to file is "fast". The recording AIS will always read as much as is available
- in the circular buffer, the recording position will immediately reach the write position
- not a problem per se, but when the user stops recording, the file may contain much more audio data than was heard (if speaker read position was behind write position)

➔ the recording position should not exceed the speaker read position

Problems and Solutions

Format Salad

- Many different audio formats
- need to be "harmonized", e.g. sample rate should not be converted
- especially with shoutcast, the input format can vary widely
- ➔ decision that input format is governing the sample rate
- ➔ there are still combinations possible that will cause an error message, e.g. 8000Hz input stream and writing to mp3 file

Future Enhancements

- Complete shoutcast implementation
- circular buffer always in encoded format to save RAM
- playlist support
- improve GUI... or better implement this technology into existing media players like jlGUI

Summary

You have learned...

- ...how to build a simple media player in pure Java
- ...audio streaming concepts
- ...about some limitations in Java Sound
- ...some tips and tricks how to overcome common problems

For More Information

- Demo application and downloads: http://www.jsresources.org/apps/radio/
- Tritonus (incl. download of MP3, ogg plug-in): http://www.tritonus.org
- Florian.Bomers@bome.com

Q&A

Time Turner Java Sound

java.sun.com/javaone/sf

Florian Bomers
bome.com