The Java Sound Internet Phone

java.sun.com/javaone/sf

Florian Bomers
Sun Microsystems, Inc.

Matthias Pfisterer
itservices pfisterer

Overall Presentation Goal

Simple VoIP with Java

Learn how to build a simple Voice over IP (VoIP) application with Java.

Explore how to leverage new features in J2SE 5.0.

Speaker Introduction

- Florian Bomers is leading Java Sound development at Sun Microsystems
- Matthias Pfisterer is an independent contractor mainly for Java technology-based projects
- Both have been programming with the Java Sound API since its very beginning
- They lead the Tritonus project – an open source implementation of the Java Sound API, and plugins.
- They founded the jsresources.org project (Java Sound Resources): open source FAQs, examples, applications.
- Today's application is available at jsresources.org

Agenda

Demo

General Architecture and program details

New Tiger Features

Problems and Solutions

Future Enhancements

Your Questions

Demo

The Java Sound Internet Phone in Action!

General Architecture

Network I

- Uses a simple TCP connection
- Not well suited:
  - delay, jitter
  - overhead
- But very simple: `InputStream`, `OutputStream`
- Connection class is abstract; UDP connection fits into architecture

General Architecture

Network II

- The active side initiates the connection
- The listener (passive side) accepts connection
- The active side sends a small header with magic, protocol version, audio format code
- The passive side responds with ACK
- From then on, only direct audio

General Architecture

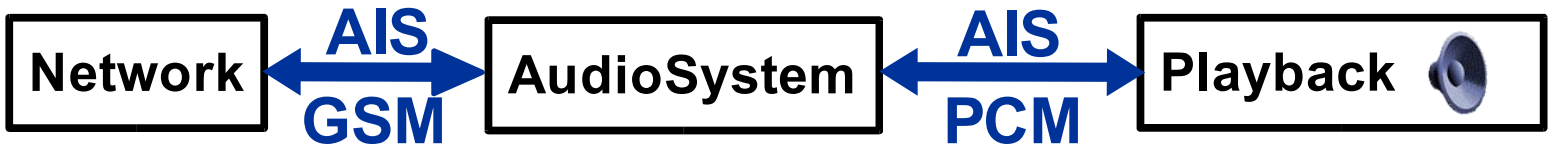
Audio I: general

- Network format:
  - possibly compressed
  - choice of GSM, phone, FM, CD quality
- Line format:
  - the format that the soundcard is opened
  - usually only PCM
- use `AudioSystem.getAudioInputStream` to convert network audio format to line format
- use new Direct Audio devices for high performance

General Architecture

Audio II: playback

- Use `SourceDataLine` for streaming playback
- receives audio data from network in an `AudioInputStream`
- converted PCM stream is read in a thread and written to the `SourceDataLine`
- use `SourceDataLine`'s buffer size for reading and writing



General Architecture

Audio III: capture

- Use `TargetDataLine` for streaming capture
- The `TargetDataLine` is wrapped in an `AudioInputStream` so that it can be converted to the network format with `AudioSystem`
- a capture thread reads from the converted `AudioInputStream` and writes it to the network connection's `OutputStream`

General Architecture

Audio IV: Mixers, Ports

- 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
- User can change the volume of an arbitrary `Port`:
  - a list of all Ports
  - a volume slider
  - does not select which output port is used for playback!

New Tiger Features

Ports

- Select input port on soundcard
- Adjust volume for input and output ports
  - gain, pan, mute, select
- Partial implementation available in J2SDK 1.4.2
- Since JDK 1.5: available on all platforms

New Tiger Features

Direct Audio

- New, additional set of Mixers
- “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

New Tiger Features

sound-properties -- default devices

- Specify system default device for `SourceDataLine`, `TargetDataLine`, ... in jre/lib/sound/properties file:

```
# use ALSA plughw:1 device as default SourceDataLine
javax.sound.sampled.SourceDataLine=#AudioPCI [plughw:1,0]
```

- Use new convenience methods:

```
// retrieve the default SourceDataLine
// with a given format:
SourceDataLine line=AudioSystem.getSourceDataLine(format);
```

New Tiger Features

New Language Features

- Static import:

```
import static org.jsresources.apps.chat.Constants.*;
```

- Generics:

```
List<String> portNames = new List<String>();
```

- Autoboxing:

```
List<Integer> controlIndex = new List<Integer>();
controlIndex.add(10); // <=> add(new Integer(10))
```

- Enhanced For:

```
List<String> getMixerNames(List<Mixer> mixers) {
  List<String> res = new ArrayList<String>();
  for (Mixer m: mixers) {
    res.add(m.getMixerInfo().getName());
  }
  return res;
}
```

Problems and Solutions: Audio

Port assignments

- Problem: now we can access all `Port`s on the system, but which `Port` really selects, e.g., the microphone?
- Which port will adjust the volume for the selected `Mixer` and `SourceDataLine`?

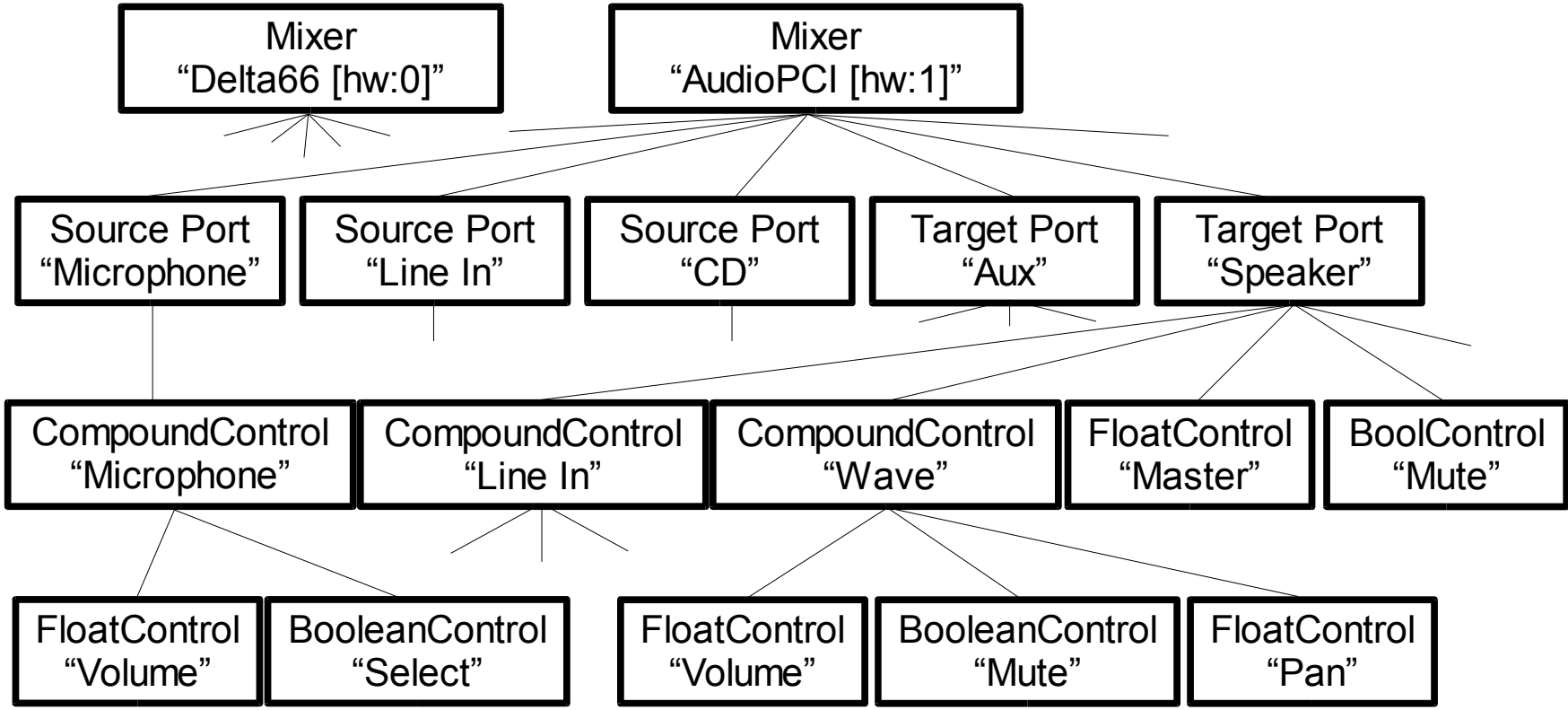
➔ Java Sound does not give access to this connection.

➔ 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: Audio

Create a list of all Ports

- Many ports per soundcard, flexible configurations possible!



Problems and Solutions: Audio

Create a list of all Ports

- Solution: use the fact that all volume and select controls will reside in a `CompoundControl`. Only use top-level CompoundControls:

```
void addPortControls(List<String> portNames,
                     Mixer mixer, Port port) {
  port.open(); // need to open port to access Controls!
  Control[] controls = port.getControls();
  for (Control c: controls) // enhanced "for" with arrays
    if (c instanceof CompoundControl)
      portNames.add(mixer.getMixerInfo().getName()
                    +": "+c.getType().toString());
}
```

```
Delta66 [hw:0]: Microphone
Delta66 [hw:0]: Line In
AudioPCI [hw:1]: Microphone
AudioPCI [hw:1]: Wave
...
```

Problems and Solutions: Audio

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: Audio

Test Mode for Microphone

- We wanted to provide a test mode: microphone is enabled and everything that is captured is played on the speakers
- Easy with the stream architecture: Use the capture `AudioInputStream` (which reads from the `TargetDataLine`) as input for the Playback class.
- The Playback class will “think” that the stream comes from the network.

Problems and Solutions: Audio

Level Meter

- We want to display the current level of the microphone and of the speaker
- `DataLine` has a method: `getLevel()` which is not implemented!
- ➔ Need to calculate the current level on our own
- ➔ for each buffer that is read from the `TargetDataLine`, calculate the maximum value and store as current level
- ➔ in a separate thread, display the current level in a `JProgressBar` every 50 milliseconds

Problems and Solutions: Audio

Level Meter: Code, 8-bit

```
protected void calcCurrVol(byte[] b, int off, int len) {
  int end = off+len;
  int sampleSize = (lineFormat.getSampleSizeInBits() + 7) / 8;
  int max = 0;
  if (sampleSize == 1) {
    // 8-bit
    for ( ; off < end; off++) {
      int sample = (byte) (b[off] + 128);
      if (sample < 0) sample = -sample;
      if (sample > max) max = sample;
    }
    lastLevel = max;
  } else if (sampleSize == 2) {
    ...16-bit next slide...
  }
}
```

Problems and Solutions: Audio

Level Meter: Code, 16-bit

```
... 16-bit:
if (lineFormat.isBigEndian()) {
  // 16-bit big endian
  for ( ; off < end; off+=2) {
    int sample = (short) ((b[off]<<8) | (b[off+1] & 0xFF));
    if (sample < 0) sample = -sample;
    if (sample > max) max = sample;
  }
} else {
  // 16-bit little endian
  for ( ; off < end; off+=2) {
    int sample = (short) ((b[off+1]<<8) | (b[off] & 0xFF));
    if (sample < 0) sample = -sample;
    if (sample > max) max = sample;
  }
}
// scale down to 0..127
lastLevel = max >> 8;
```

Problems and Solutions: Audio

“Mute” button

- We want to provide a mute button -- may need to cover up “private” noises :)
- Could use Port's mute or Volume controls
  - but are we sure that it's the right port?
- ➔ Implement a soft-mute: just zero the buffer if muted
- ➔ this way we're absolutely sure that it is really muted

Problems and Solutions: Audio

“Mute” button: Code

```
... capture code:
  int ret = line.read(b, off, len);
  if (isMuted()) {
    muteBuffer(b, off, ret);
  }
...

protected void muteBuffer(byte[] b, int off, int len) {
  int end = off+len;
  int sampleSize = (lineFormat.getSampleSizeInBits() + 7) / 8;
  byte filler = 0;
  if (sampleSize == lineFormat.getChannels()) {
    // 8-bit has -128 as silence
    filler = -128;
  }
  for ( ; off < end; off++) {
    b[off] = filler;
  }
}
```

Problems and Solutions: Network

Incomplete `read()`

- Calling `read()` on a `Socket`'s `InputStream` may return less bytes than requested
- Solution: `DataInputStream.readFully()`

```
Socket sock = ...;
InputStream inStream = sock.getInputStream();
byte buffer = new byte[4];

// may read any between 0 and 4 bytes
inStream.read(buffer);

// guaranteed to read 4 bytes
DataInputStream dataInStream =
        new DataInputStream(inStream);
dataInStream.read(buffer);
```

Problems and Solutions: Network

TCP latency

- Original round-trip delay: 1,5 seconds
  - Delay end-to-end (one direction): about 750 ms
- Solution: set send buffer size and receive buffer size for sockets
- With 1024 bytes: delay barely noticable
- Size in bytes should depend on audio format

```
void setSocketOptions(Socket sock) {
  sock.setNoDelay(false);  // delayed ACK improves
                           // performance
  sock.setSendBufferSize(1024);
  sock.setReceiveBufferSize(1024);
}
```

Problems and Solutions: Network

Connection detection

- "Active" side: plain Socket
  - Initiates connection
- "Passive" side: ServerSocket
  - Waits for incoming connections

```
Socket commSock; // communication socket
void connect(INetAddress addr) {
  commSock = new Socket(addr, PORT); // tries to connect
  setSocketOptions(commSock);
}

ServerSocket servSock = new ServerSocket(PORT);
void listen() {
  commSock = servSock.accept(); // blocks until conn. attempt
  setSocketOptions(commSock);
}
```

Problems and Solutions: Network

Disconnect detection

- One side closes its communication socket to initiate a disconnect
- The peer detects it by receiving exceptions when reading from or writing to the socket
  - Not yet implemented cleanly

Problems and Solutions: Network

Two users on the same machine

- Only one process can listen for connections on a specific port numer
  - Two instances need to use different port numbers
- Solution: port number as configuration option

Summary

You have learned...

- ...how to build a simple VoIP application in pure Java
- ...how to use Tiger's new language and sound features
- ...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/chat/
- Tritonus (incl. download of GSM plug-in): http://www.tritonus.org
- Florian.Bomers@sun.com
- Matthias.Pfisterer@web.de

Q&A

JavaOne℠

Sun's 2004 Worldwide Java Developer Conference℠

The Java Sound Internet Phone

java.sun.com/javaone/sf

Florian Bomers
Sun Microsystems, Inc.

Matthias Pfisterer
itservices pfisterer

Sun microsystems