JavaOne℠

Sun's 2001 Worldwide Java Developer Conference℠

Applet Programming With the Java™ Sound API

Florian Bomers
Software Engineer
bome.com

Matthias Pfisterer
Software Engineer
itservices Pfisterer

Overall Presentation Goal

Learn how to create applets that use the Java™ Sound API for recording and playing audio data

Get an overview of the possibilities and limits when deploying sound enabled applets

Learning Objectives

As a result of this presentation, you will know how to

- Play and record streamed audio data
- Encode and decode to and from GSM
- Transfer sound data to and from a server
- Handle applet and Java Sound API security management
- Sign applets

Speakers' Qualifications

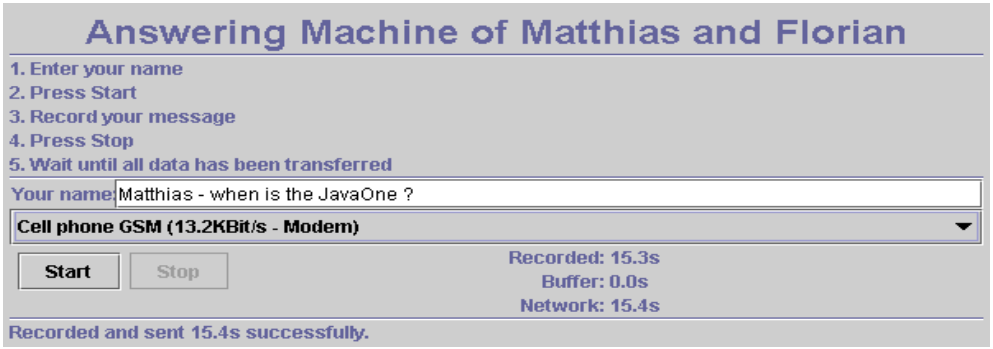
- Florian Bomers is freshly hired at Sun Microsystems in the Java Sound department
- Matthias Pfisterer is an independent contractor mainly for Java technology-based projects
- Both have been programming with the Java Sound API since its very beginning
- Both are leading the Tritonus project—an open source implementation of the Java Sound API

Presentation Agenda

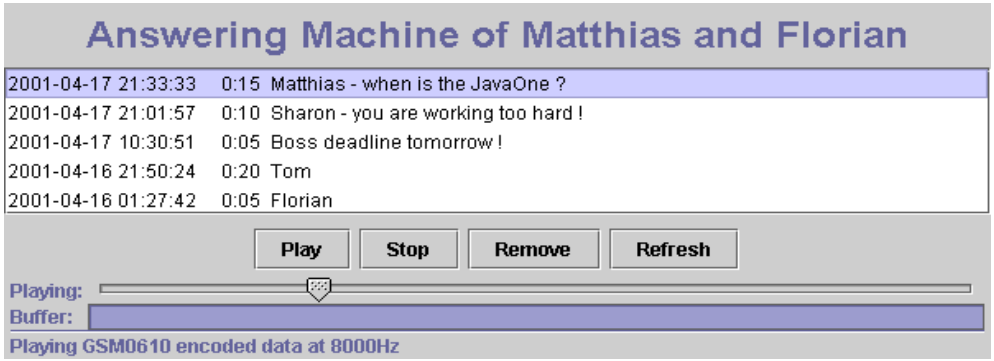
- Demo of the example application: a web-based answering machine
- General architecture and program details
- Deploying the applets
- Problems and solutions
- Your questions

Answering Machine Demo

- Caller:



- Owner:



General Architecture

Caller Data Flow

- Applet records audio data from soundcard
- Applet sends it to the server
- Server receives audio data
- Server saves data in a file

General Architecture

Owner Data Flow

- Server reads audio data from file
- Server sends it to the applet
- Applet receives audio data from server
- Applet plays back the audio data

General Architecture

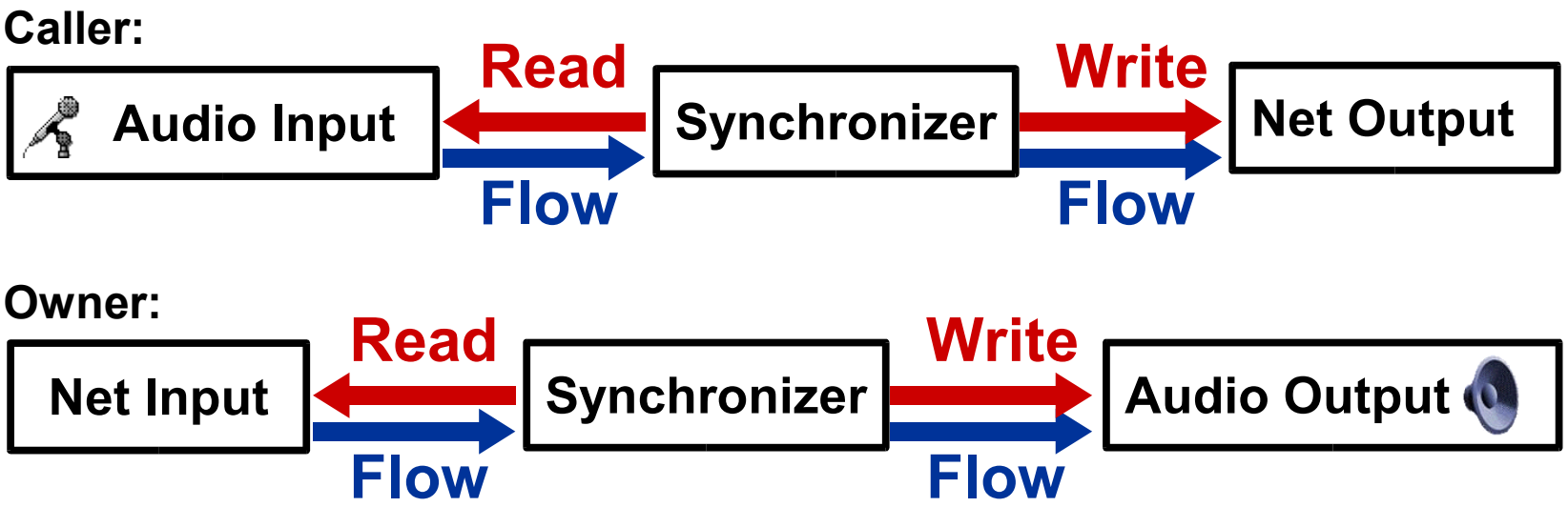
Streams

- Audio data flows in streams
- Recording uses a subclass of **`InputStream`** that reads from a **`TargetDataLine`**
- Net i/o is done with **`InputStream`** / **`OutputStream`** provided by the **`java.net.URLConnection`** class
- Playback uses a subclass of **`OutputStream`** that writes to a **`SourceDataLine`**

General Architecture

Streams: Synchronizer

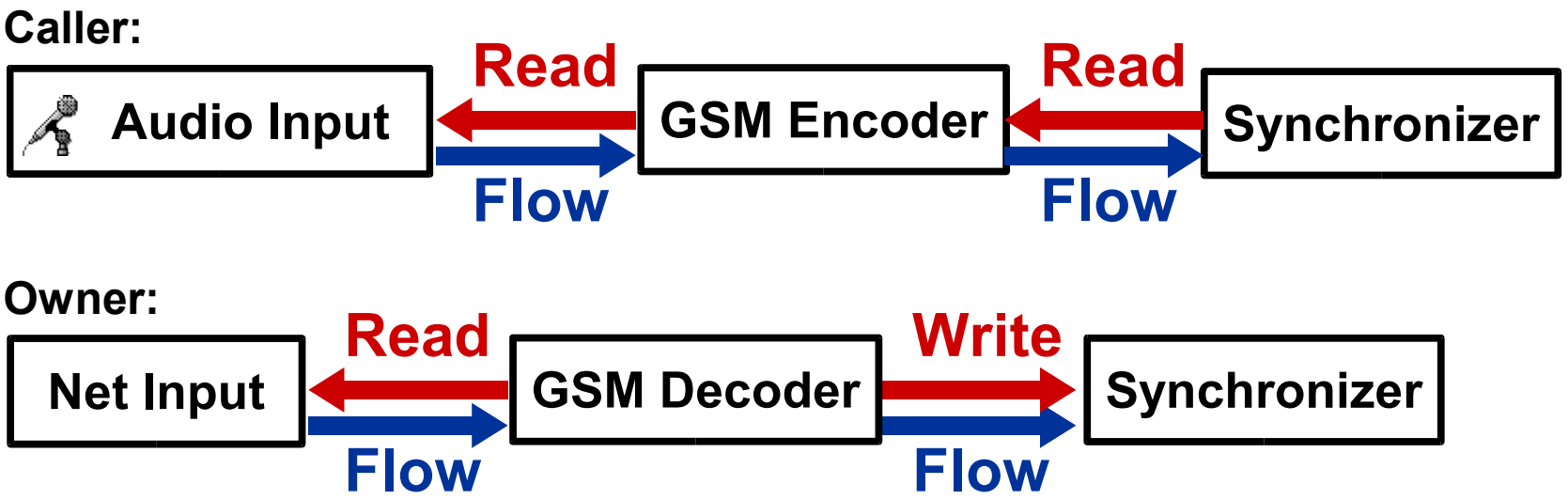
- The synchronizer reads from an **`InputStream`** and writes to an **`OutputStream`**
- It changes flow from pull to push



General Architecture

Stream Cascading

- Input streams are cascaded to process data
- One stream reads from the underlying stream and returns the processed output



General Architecture

Net Communication

- A standard CGI program for a web server
- Methods implemented by the server CGI: GET, PUT, LIST, REMOVE
- The CGI program saves uploaded messages to files
- Could easily be done as Servlet
- We also implemented a stand-alone server program

Program Details

Synchronizer

- The “heart” of audio data flow
- Runs in its own thread

```
// simplified...
public void run() {
  while (true) {
    int bytesRead =
        inStream.read(
          buffer, 0, buffer.length);
    outStream.write(buffer, 0, bytesRead);
  }
```

(org.jsresources.am.audio.Synchronizer)

Program Details

Caller: Overview

- Open a **`TargetDataLine`**
- Get an **`AudioInputStream`** from it
- Get a converted **`AudioInputStream`** in GSM format from it
- Open network connection to server and get its **`OutputStream`**
- Connect **`AudioInputStream`** and network output stream to **`Synchronizer`**

Program Details

Caller: open TargetDataLine

```
AudioFormat format =
      [signed PCM, 8000Hz, 16bit, mono]

DataLine.Info info = new DataLine.Info
      (TargetDataLine.class, format);

TargetDataLine line =
      (TargetDataLine)
         AudioSystem.getLine(info);

line.open(format, bufferSizeInBytes);
```

```
(org.jsresources.am.audio.AudioCapture)
```

Program Details

Caller: get AudioInputStream

```
AudioInputStream pcmAIS =
            new AudioInputStream(line);
```

- `AudioInputStream` is an `InputStream` with
  - Audio format definition
  - Optional length definition

(org.jsresources.am.audio.AudioCapture)

Program Details

Caller: convert to GSM I

- Codecs are plug-ins to the Java™ Sound API using the Extension Mechanism (Service Providers)
- They work by cascading an **`AudioInputStream`**
- Here we use the GSM 06.10 plug-in released by Tritonus
- GSM provides lossy compression well adapted for low bitrate speech data

Program Details

Caller: convert to GSM II

```
AudioFormat.Encoding gsmEncoding =
     Encodings.getEncoding("GSM0610");

AudioInputStream gsmAIS =
     AudioSystem.getAudioInputStream(
          gsmEncoding, pcmAIS);
```

- **`Encodings`** is a utility class of Tritonus (bundled with GSM plug-in)
- **`AudioSystem`** retrieves the GSM codecs from the list of installed plug-ins

`(org.jsresources.am.CallerClient)`

Program Details

Caller: Network Connection

- Network may be slower than audio data coming in

➔ network connection is buffered

  - The network output stream is cascaded in an **`OutputStream`** that queues all data written to it
  - In a thread, it writes all data to the network output stream

`(org.jsresources.am.BufferedSocketOutputStream)`

Program Details

Caller: Using the Synchronzer

```
Synchronizer sync = new Synchronizer(
      gsmAIS,                    // input stream
      socketOutputStream, // output stream
      audio.getBufferSizeInBytes());
// Start audio (TargetDataLine)
audio.start();
// Start network (thread)
socketOutputStream.start();
// Start Synchronizer (thread)
sync.start();
```

(org.jsresources.am.CallerClient)

Program Details

Onwer: Overview

- Open network connection to server and get its **`InputStream`**
- Create **`AudioInputStream`** (GSM) from it
- Get a converted **`AudioInputStream`** in PCM format from it
- Open a **`SourceDataLine`** in a class that subclasses OutputStream **(`AudioPlayStream`)**
- Connect **`AudioInputStream`** and **`AudioPlayStream`** to Synchronizer

Program Details

Onwer: Details

- Code is analogous to Caller
- Audio data is buffered and stored for later usage (i.e., rewind)

Deploying the Applets

Overview

- Write GUI and Applet classes*)
- Package the classes in a jar
- Create HTML pages
- Respect security!
- Signing (optional)

*) Not handled here

Deploying the Applets

Create Jar Archive

- The applets need the GSM plug-in
- The **`Class-Path`** header in a jar manifest allows to download additional packages
- manifest.mf:

```
Manifest-Version: 1.0
Class-Path:
tritonus_gsm.jar
```

- Creation of jar:

```
jar cmf manifest.mf am.jar *.java org
```

Deploying the Applets

Create HTML Pages: Standard Approach

```
<APPLET CODE =
"CallerClientApplet.class"
        ARCHIVE = "am.jar"
        WIDTH = "600"
        HEIGHT = "250">
    <PARAM NAME = "server"
           VALUE ="/cgi-bin/am.cgi">
</APPLET>
```

- Uses JVM* of browser
- Not many browsers have a JDK1.3 JVM (exception: Netscape 6)

*As used in this presentation, the terms “Java™ virtual machine” or “JVM™” mean a virtual machine for the Java™ platform

Deploying the Applets

Create HTML Pages: Use Java 2 Plug In Technology

- Use *HTMLConverter* to make the HTML page use Java 2 Plug-In technology
  - Creates **`<object>`** tag for Internet Explorer (IE)
  - Creates **`<embed>`** tag for Netscape
- Netscape ignores the **`<object>`** tag, while IE ignores **`<embed>`**

Deploying the Applets

Security: Overview

- By default, applets are not allowed to record audio data (eavesdropping)
- The Java 2 platform offers a flexible concept of assigning fine-grained permissions
- Security is handled on the client

Deploying the Applets

Security: Overview

- Each permission is bound to a permission object (e.g. **`java.io.FilePermission`**)
- A permission may have one or more target names: (e.g., “read”, “write”, or “*” for all)
- Once a protected method is accessed, the JVM checks if the permission is granted (e.g., trying to write to a file)
- If not, throws an instance of **`java.security.AccessControlException`**

Deploying the Applets

Security: Policy Files

- Permissions are set in 2 files:
  - system policy file in
    `JAVAHOME/lib/security/java.policy`
    `JAVAHOME e.g.:`
    `C:\Program Files\JavaSoft\JRE\1.3.0_02`
  - user policy file in
    `USERHOME/.java.policy`
    `USERHOME e.g.:`
    `C:\Documents and Settings\florian`

Deploying the Applets

Security: Setting Permissions in File

- For recording audio, the permission **`javax.sound.sampled.AudioPermission`** with value "**`record`**" is needed
- Create a user policy file with this content:

```
grant {
  permission
    javax.sound.sampled.AudioPermission
    "record";
};
```

- or...

Deploying the Applets

Security: Setting Permissions With Policytool

- ...use the graphical frontend *policytool*:
- Click on *Add Policy Entry, Add Permission*
- Enter *Permission:*
  `javax.sound.sampled.AudioPermission`
- Enter *Target Name:*
  `record`
- *OK*, *Done*, *File*|*Save*
- More user-friendly than directly editing the policy file

Deploying the Applets

Security: What is Signing?

- Using cryptographic algorithms to
  - Assure the identity of the signer
  - Assure the integrity of the code
- But it does not
  - Give privacy (no encryption)
  - Provide protection against malicious code/DOS attacks/etc

Deploying the Applets

Signing: What is a Certificate?

- My public key, signed by a CA (certification authority)
- CA's act as Trusted Third Party
- CA's are, e.g., VeriSign, Thawte, Entrust
- A certificate can be validated by verifying its signature (using the CA's public key)
- X.509 certificates are used e.g., for signing applets or for the SSL protocol (https)

Deploying the Applets

Signing: How to Sign an Applet

1) Buy a certificate from a CA
2) Make it available locally (import it)
3) Sign the jar file:

```
jarsigner am.jar myname
```

4) Verify the signature (optional):

```
jarsigner -verify [-verbose] am.jar
```

Deploying the Applets

Signing: Signed Applets

- When a signed applet is loaded with Java Plug-In technology, a security dialog pops up
  - The user can inspect/verify the certificate
  - The user can grant “All permissions” (i.e., fine grained permissions are not possible)
- Silent failure for invalid certificates (and “All permissions” is not granted)
- A granted certificate is cached by the plug-in and all applets signed by that certificate are automatically granted “All Permissions” (see Java™ plug-in control panel)

Problems and Solutions

Problem: Plug-ins in Applets

- Since the JDK™ 1.3.0_01, applets may not install a Service Provider Extension (like the GSM plug-in) over the Internet
- Even that the GSM classes are accessible (due to **`Class-Path`** header in manifest), the GSM plug-in is not installed in **`AudioSystem`**

Problems and Solutions

"Problem": Plug-ins in Applets

- Instantiate the GSM Service Provider directly:

```
// GSMFormatConversionProvider in package
// org.tritonus.sampled.convert.gsm
AudioInputStream gsmAIS = new
  GSMFormatConversionProvider().
```

- Not a nice solution !
- Better with Java Web Start software

```
(org.jsresources.am.audio.AMAppletWorkaround)
```

Problems and Solutions

Problem: Restricted AudioFileFormat

- We would have liked to use a standard file format and use **`AudioSystem`** methods for reading/writing
- The “caller name” must be included in the header
- E.g. field “description text” in AU files or “list chunk” in WAVE files
- **`AudioFileFormat`** does not provide fields for additional information of a file

Problems and Solutions

"Solution": Own File Format

- We defined our own file format
- It is like AU
- Not nice as we have to "re-invent the wheel"

(org.jsresources.am.audio.AMMsgFileTool)

Problems and Solutions

Problem: Buffered URLConnection

- When streaming to or from the web server, **`URLConnection`** queues the data until the transfer is finished
  - Uses much memory for long recordings/messages
  - Prevents simultaneous transfer over the Internet while recording or playing

➔ Not suitable for this application

Problems and Solutions

Solution: Own URLConnection

- An own class that communicates with the web server
- Not nice, as again we have to create a class that already exists in the JVM
- New problem: HTTP/1.0 does not allow upload of unknown length (Content-length header must be set); better: use HTTP/1.1

`(org.jsresources.am.net.AMHttpClientConnection)`

Future Enhancements

- Make it a Java™ Web Start software-based application
- Caller: Possibility to add a text message
- Owner: Access restriction (password)
- Owner: Multi-user
- Server: As a servlet
- Server: Use a database instead of files

Summary

- We showed how to
  - Stream audio data in GSM format to and from a web server
  - Deploy applets for different VM's
  - Deal with security restrictions of applets
  - Create signed applets
  - Overcome limits of the current JDK release

Related Sessions and BOFs

- TS-541: Developing Advanced Multimedia Applications with Java™ Technology
  - Friday June 8, 9:45 AM, Moscone Center—Hall B

Reference

- Demo application download and docs:
  http://www.jsresources.org/am/
- Tritonus (incl. download of GSM plug-in):
  http://www.tritonus.org
- Java™ 2 plug-in homepage (incl. HTML Converter download):
  http://java.sun.com/products/plugin/
- JDK™ 1.3 software security guide:
  http://java.sun.com/j2se/1.3/docs/guide/security/
- Java™ Web Start software™:
  http://java.sun.com/products/javawebstart/index.html

florian @ bome.com
Matthias.Pfisterer @ web.de

JavaOne℠

Sun's 2001 Worldwide Java Developer Conference℠

Q&A

JavaOne℠

Sun's 2001 Worldwide Java Developer Conference℠