Productizing Telephony and Audio in a GNU/Linux (Sailfish OS) Smartphone

Martti Piirainen (martti.piirainen@tieto.com)
Tieto Product Development Services
Agenda

• Hardware / software stack overview
• Audio
  – What we did and why
  – Some implementation details
• Telephony
  – What we did and why
  – Some implementation details
About the speaker

● Martti Piirainen
  – Cellular telephony expert (from modem to UI)
  – 15 years in SW development, Linux mobile devices since 2010

● Tieto Product Development Services
  – R&D in communications and embedded technologies
  – Part of Tieto, 14000 employees, headquarters in Finland
  – More at www.tieto.com/pds

● This presentation covers work done by Tieto for Jolla
Hardware Stack

- **“Jolla”** smartphone
- **Qualcomm MSM8930 SoC** (from “Snapdragon S4” family)
  - incl. dual-core Krait ARM CPU
  - incl. GSM/UMTS/LTE cellular modem
  - incl. multimedia accelerators
Software Stack

- **Sailfish OS**
- UI layer mostly done in **QML**
- Middleware based on **Nemo Mobile** project
- Based on **Mer** Linux distribution
- **Android™** BSP plus some **libhybris** magic
Audio Use Cases

• Play music, video, anything with sound in it
  – Media player, web browser, 3rd party apps, ...

• System / feedback sounds

• Cellular voice calls
  – Two-way speech
  – Ringtone
  – Generate various signalling sounds

• Volume control / muting
Audio Use Cases, cont'd

- Output devices
  - Speaker, earpiece, wired or Bluetooth headphones

- Input devices
  - Built-in microphone, wired or Bluetooth headset microphone

- Audio routing
  - Which sound input (software process or physical device) is connected to which sound output?

- Audio policy
  - Which apps / services are allowed to use which resource? Example: Ringtone takes precedence over media player
  - Also, automatic routing changes. Example: Plug in headset ==> route audio to headset & adjust volume
Audio Routing & Policy Stack

- GStreamer & other clients
- PulseAudio & OHM
- module-droid
- Android Audio HAL
- ALSA drivers
- Qualcomm Audio (& Bluetooth & Modem) HW
Some of our Work in Audio

• New PulseAudio modules
  – `module-droid-{card,source,sink,keepalive}`
  – Use Android HAL (via libhybris) for routing
  – Media audio routes through these streams
  – Cellular audio and Bluetooth audio streams stay in the SoC side ...
  – ... but we control routing, volume, muting, etc.
Some of our Work in Audio, cont'd

• Implement all resource policy in Sailfish
  – Based on Maemo / MeeGo legacy

• Productize:
  – Tune the configurations (e.g. volume levels, priorities)
  – Test, bugfix, rinse, repeat

• Alternative approach: We might have used PulseAudio
  ALSA-modules and ALSA Use Case Manager
  – Also a lot of work, esp. UCM porting
  – Less portable
PulseAudio and Audio HAL talk via C function calls

Initialization snippet from pulseaudio-modules-droid/src/droid/droid-util.c:

```c
hw_get_module_by_class(AUDIO_HARDWARE_MODULE_ID, module->name, (const hw_module_t **) &hwmod);

if (!hwmod) {
    pa_log("Failed to get hw module %s.", module->name);
    goto fail;
}
```

```c
ret = audio_hw_device_open(hwmod, &device);
```

from libhybris (hardware.c)

from Audio HAL (audio.h)
Audio routing example: Speaker during call

• During a voice call, enable the Integrated Hands-Free
  – i.e. change downlink speech audio routing from earpiece to speaker

• Sequence:
  Voicecall UI
  ==> OHM
  ==> Dependency Resolver
  ==> PulseAudio policy module
  ==> core PulseAudio
  ==> PulseAudio droid-sink module
  ==> Android Audio HAL
Audio routing example cont'd

Snippet from pulseaudio-modules-droid/src/droid/droid-sink.c:

```c
pa_snprintf(tmp, sizeof(tmp), "%s=%u;",
            AUDIO_PARAMETER_STREAM_ROUTING, routing);

pa_log_debug("set_parameters(): %s (%#010x)",
            tmp, routing);

pa_droid_hw_module_lock(u->hw_module);

u->stream_out->common.set_parameters(
            &u->stream_out->common, tmp);

pa_droid_hw_module_unlock(u->hw_module);
```

from Audio HAL (audio.h)
Telephony

- The “phone” part of “smartphone”
- Everything around Cellular (GSM / WCDMA / LTE) connectivity
- Most visibly voice calls, text messaging, packet data ...
- ... but also many more obscure and “legacy” things
- The modem implements the “hard” protocol-level problems, but *modem interfaces* are still massive beasts
Telephony, cont'd

- Linux Telephony daemon: oFono
  - Design principle: hide unnecessary complexity from the phone UI
  - Happy 5th Birthday, oFono!
Telephony Stack

Telepathy & other clients

oFono

RIL (Java)

Android RIL daemon

QMI drivers

Qualcomm Modem
(& Audio & Bluetooth) HW
Some of our Work in Telephony

• Extend existing oFono rilmodem driver
  – https://github.com/rilmodem/ofono, used by **Ubuntu Touch**
  – Basic use cases of voice call / SMS / packet data were already implemented by Canonical

• Fix what's broken and add what's missing
  – https://github.com/nemomobile-packages/ofono, used by **Sailfish OS**
  – Examples following:

• **SIM**-related things
  – Security handling (PIN, PUK), phonebook access
  – SIM Toolkit. “Legacy” but requested by operators. Example: A mobile authentication service
Some of our Work in Telephony, cont'd

- Advanced **call handling**
  - Hold/resume, multiparty
  - Emergency calls always possible. *Regulatory requirement in some markets.*
  - Signalling tones
- **SMS** improvements
  - Delivery report handling
  - Text encodings. *Example: SMS containing advanced Unicode (emoticons as surrogate pairs)*
- **Network** handling
  - Operator selection (automatic / manual), roaming behaviour, user preferences
  - Show network name / Service Provider name. *Important for operators' branding.*
  - Flight mode
Some of our Work in Telephony, cont'd

- **Supplementary Services**
  - Call Forwarding / Waiting / Barring
  - CLIR (show / hide my number), USSD

- **Settings provisioning**
  - Packet data and MMS. Pre-configured and Over-The-Air

- **Modem power management** based on system activity state

- “OEM Raw” **extension API** for proprietary modem requests

- **Fixes in telepathy-ring**, the port from csd to oFono was somewhat unfinished
Some of our Work in Telephony, cont'd

• **Productize:**
  - Prioritize requirements from product mgmt, operators, end users (*we just couldn't do everything at once*)
  - Don't just do the “happy cases”. *Example: Many requests can be rejected by the network and/or SIM*
  - Test, bugfix, rinse, repeat
  - Including surprises from field testing
  - Utilizing network and SIM card simulators

• **Alternative approach:** We might have extended oFono's QMI driver
  - RIL API is on a slightly higher level of abstraction, increases portability, hides modem-specific quirks
  - RIL implementation is ”battle-hardened” in Android devices
oFono and RIL talk via UNIX socket and messages

Initialization snippet from ofono/gril/gril.c:

```
#define RILD_CMD_SOCKET  "/dev/socket/rild"
addr.sun_family = AF_UNIX;
strncpy(addr.sun_path, RILD_CMD_SOCKET,
  sizeof(addr.sun_path) - 1);
if (connect(sk, (struct sockaddr *) &addr,
  sizeof(addr)) < 0) { ... }
```
ofono example: Call Waiting status

- Set / Query the status of the “Call Waiting” supplementary service (a.k.a. “knocking”) to / from the network

- Client calls ofono via D-Bus:
  - Object path /ril_0
  - Set the “call waiting” status:
    Method org.ofono.CallSettings.SetProperty, arguments (string: "VoiceCallWaiting", variant: string: "enabled")
  - Get the “call waiting” status:
    Method org.ofono.CallSettings.GetProperties
oFono example cont'd, oFono behavior

- The D-Bus request is handled by oFono's call-settings atom
  - checks if busy with a pending request, and argument validity
  - checks availability of a modem driver implementing this

- Request is forwarded to rilmodem call-settings driver
  - Driver constructs a request message
  - gril plug-in sends it to RIL
    - which does the actual modem and network request
  - Driver parses the response message
  - Callback to oFono core, reply to D-Bus client
oFono example cont'd, RIL messages

Request message, oFono ==> RIL

```
0000001023000000e8010000010000000000000000
```

- msg length
- serial number
- RIL_REQUEST_QUERY_CALL_WAITING, from ril.h
- service class “all”

Response message, RIL ==> oFono

```
0000001800000000e80100000000000000020000000100000000
```

- "enabled" ... for "voice"
Achievements Summary

- Using Android HAL and RIL was **speeding up** our product development
  - From first hacks on target hardware to commercial launch in a few months
  - CE and Bluetooth HFP certified
  - A beautiful phone with happy users
Acknowledgements

• Warm thanks to:
  – Everyone at Jolla
  – Canonical rilmodem team
  – oFono, PulseAudio, libhybris maintainers and contributors
Repositories in GitHub.com

mer-packages/pulseaudio
mer-hybris/pulseaudio-modules-droid
nemomobile/pulseaudio-modules-nemo
nemomobile/pulseaudio-policy-enforcement
nemomobile/ohm
nemomobile/ohm-plugins-misc
nemomobile/libdres-ohm
nemomobile/ohm-rule-engine
nemomobile/policy-settings-common
nemomobile/tone-generator
nemomobile/telepathy-ring
nemomobile/provisioning-service
nemomobile-packages/ofono

PulseAudio + modules
Audio Policy
Telephony