Telephony Framework in Tizen 2.1: What’s new?

Harish Bishnoi, Philippe Nunes
Agenda

- Introduction
- Architecture
- Tizen2.1 Features
- Call Flows
- Porting Telephony
- Future work
Introduction
Introduction | Feature Overview

- Offers rich functionalities
  - Voice Call and Messaging services
  - Supplementary services, USSD
  - SIM, Phonebook, SAT
  - Packet and A-GPS services
  - Network services (LTE, 3G/2G)

- Plug-in based Architecture
  - Modem Agnostic
  - Flexible and easily customizable as per OEM needs
Introduction | Feature Overview

● Commercial ready
  ● Verified on Ref.Device-210 and Ref.Device-PQ
  ● Verified on Intel Medfield device

● License
  ● Apache License Version 2.0

● Reference plugin available
  ● AT commands (3GPP 27.007/ 27.010) based plugin
Architecture
Architecture | Definitions

- **Core Objects**
  - Executable components of a Telephony Module (SIM, CALL, SS, etc.).
  - Bundle of functions and supportive database information designated to the Module.

- **Template Core Objects**
  - Non-Executable components, but result in executable Core Objects when cloned.

- **Hardware Abstraction Layer (HAL)**
  - Abstracts the communication channel with modem.

- **Hooks**
  - Mechanism to tap Requests/Notifications of other Telephony Modules of interest.
Architecture | Telephony Components

The major components of Telephony are –

- **Telephony libraries**
  - Telephony API (TAPI) library
  - Core Telephony library

- **Telephony Plug-ins**
  - Communicator plug-ins
  - Modem plug-ins
  - Modem Interface plug-ins (HAL)
  - Free Style plug-ins

- **Telephony Server**
Telephony API library (or simply TAPI) is a standardized interface provided to applications to interact with Telephony over DBUS. TAPI is provided as libslp_tapi and executes in application’s context. TAPI provides Sync and Async APIs.
Core Telephony library (or simply libtcore) provides an API framework for Telephony components to inter-work.

It is provided as *libtcore* package.
Tizen 2.1 Features
Tizen2.1 Features | Design consideration

- Optimize the support of various types of modem architectures from same vendor.
- Modem plug-in needs to be **hardware agnostic**.
- Modem Interface plug-in is the adaptation layer between the telephony framework and the hardware of your target device.
- Avoid code duplication by introducing Core object templates and operations over-riding.
Tizen2.1 Features | Core Object cloning

- Core Objects can be cloned from ready-to-use Template Core Objects
- Template Core Objects are stored in Server
- Cloned Core object is Executable
Tizen2.1 Features | Over-riding operations

- Operations over-riding enables Vendors to customize the Core Objects
- Over-riding can vary from 0 – 100 %
- Callbacks for notification(s) can also be over-ried
Tizen2.1 Features | CMUX support

- **3GPP TS 27.010** – Terminal Equipment to Mobile Station (TE-MS) multiplexer protocol is supported
- Currently supports Basic mode operation
- Enables different Core Objects (CALL, SIM, SMS, etc) to interact with Modem concurrently through multiple CMUX channels
Tizen2.1 Features | Internal CMUX

- **Internal CMUX uses CMUX feature of `libtcore`**
- **Each Logical HAL has a dedicated CMUX channel**
- **Logical HALs can be shared between Core Objects**
- **The HAL assignment to Core Objects is done by the mapping table**
The MUX driver is an open source implementation (N_GSM) of the 3GPP 27.010.

Multiple Physical HALs exist, each has a dedicated Kernel CMUX channel.

Physical HALs can be shared between Core Objects.
Tizen2.1 Features | Hardware agnostic Modem plug-in

- Same Modem plug-in can operate across various types of modem architectures (from same vendor)
- Hardware specific adaptation are required ONLY in Modem Interface plug-in
Call Flows | Boot-up sequence

- **Server**
  - **Load Plug-ins**
    - **Basic plug-in folder**
    - **Add Plug-in item**
  - **Init**
  - **Call Init**

- **Communicator Plugin**
  - **Load**
  - **Create communicator**
  - **Init**
  - **Call Init**

- **Template CO plug-in**
  - **Load**
  - **Template CO**
  - **Add Template CO**
  - **Call Init**

- **Modem Interface**
  - **Init**
  - **Call Init**
Call Flows | Application request sequence

- **TAPI**
  - Creates User Request

- **Communicator**
  - Dispatch User Request

- **Server**
  - Find the Plug-in
  - Find the Core object

- **Modem plug-in**
  - Core Object function dispatcher
  - Enqueue Pending

- **Modem Interface**
  - HAL Send Request data

- **Modem**
  - App Req

- **Core Object**
  - Dispatcher
Call Flows | Solicited Response sequence

TAPI

Invoke Application callback for async answer

Communicator

Prepare and send the response

Server

Send response to communicator

Modem plug-in

Response Callback
- Parse response
- Prepare UR response

Modem Interface

Noti / Resp?

Pop pending request from the HAL queue

Response data

Emit pending response

Modem

Send response to communicator

Prepare and send the response
Call Flows | Unsolicited Response sequence

- **TAPI**
- **Communicator**
- **Server**
- **Modem plug-in**
- **Modem Interface**
- **Modem**

1. Prepare and send the notification
2. Invoke callback for event
3. Dispatch notification
4. Invoke Hook functions
5. Emit unsolicited message
6. Unsolicited result code

**Communicator**

- Invoke Notification Callback Registered by Core Objects

**Modem**

- Noti/Resp?
Porting Telephony
Porting Telephony | Implementing plug-ins

- All telephony plug-ins mandatorily need to provide a descriptor structure

<table>
<thead>
<tr>
<th>Descriptor Structure</th>
<th>Description of structure elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct tcore_plugin_define_desc {</td>
<td>Structure referred by Telephony Server to load, initialize, and unload the Plug-in.</td>
</tr>
<tr>
<td>gchar *name;</td>
<td>Name of Plug-in</td>
</tr>
<tr>
<td>enum tcore_plugin_priority priority;</td>
<td>Initializing priority of the Plug-in</td>
</tr>
<tr>
<td>int version;</td>
<td>Plug-in version</td>
</tr>
<tr>
<td>gboolean (*load)();</td>
<td>Plug-in load function reference</td>
</tr>
<tr>
<td>gboolean (*init)(TcorePlugin *);</td>
<td>Plug-in init function reference</td>
</tr>
<tr>
<td>void (*unload)(TcorePlugin *);</td>
<td>Plug-in unload function reference</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
- OEM vendors can customize each and every Telephony plug-in as per their needs.
- It is **NOT** mandatory that all the plug-ins need to be changed to support a specific hardware.
Porting Telephony | Vendor plug-in requisites

- **Modem Interface plug-in**
  - Establish connection to modem, additionally if required setup CMUX (either internal or Kernel)
  - Create HALs (Logical and/or Physical)
  - Register new modem into server
  - Assign Core Object types to HALs (logical or physical) and update Mapping Table with the corresponding assignments
  - Request Server to load modem specific Modem plug-in

- **Modem plug-in**
  - Clone and initialize the Core Objects according the Mapping Table
  - Override the operations and callbacks (if required)
  - Start Telephony functionalities
Future work
Future work

- USB data dongle support
- VoLTE support
- CDMA modem support
- Multi-SIM