Tizen Telephony

Jongman Park

Contents

- Introduction
- Architecture
- Components
- Operation flow
- Developing plug-in

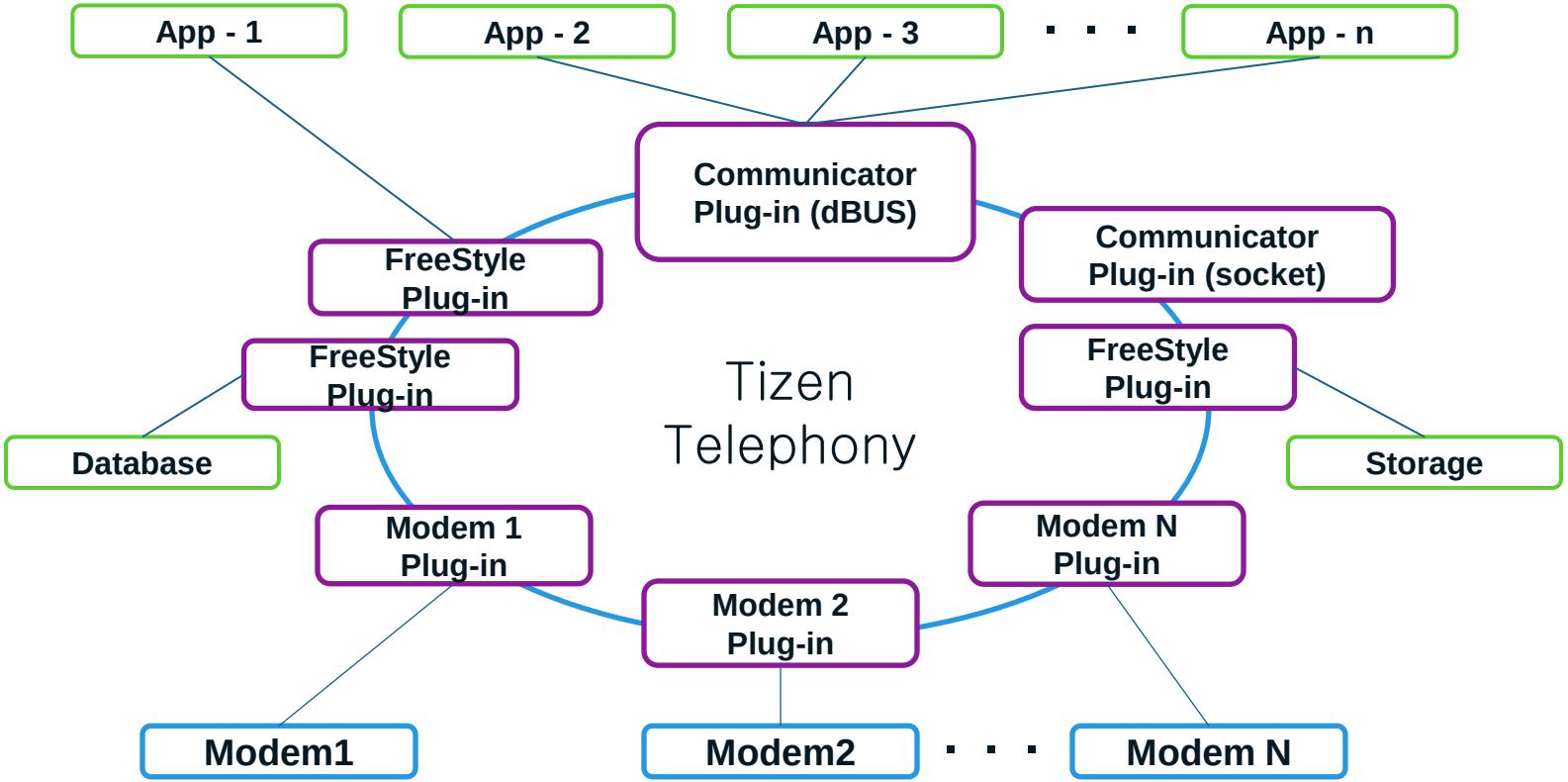
Introduction

- **Telephony stack is ready for commercialization**
  - It is a proven qualified stack with modem chip vendors in industry
  - Full compliant telecommunication functionalities
    - SIM, SIM phonebook, SIM application toolkit
    - Network registration, Voice/Video call service, Managing SMS/CBS
    - Supplementary services
    - Packet service
  - It already supports well-defined interface with connman.

- **Benefits for commercialization**
  - Flexible plug-in architecture
    - Inter-process communication plug-in
    - Modem interface plug-in
  - Keeping maintenance for commercialization readiness.
    - GCF certification
  - Can be well customized for various carrier requirements and manufacturer's proprietary requirements
    - Various carrier requirements can be easily customized with plug-in capsules
    - Manufacturers do not need to have obligation to open their proprietary implementation

*GCF : Global Certificate Forum

Architecture

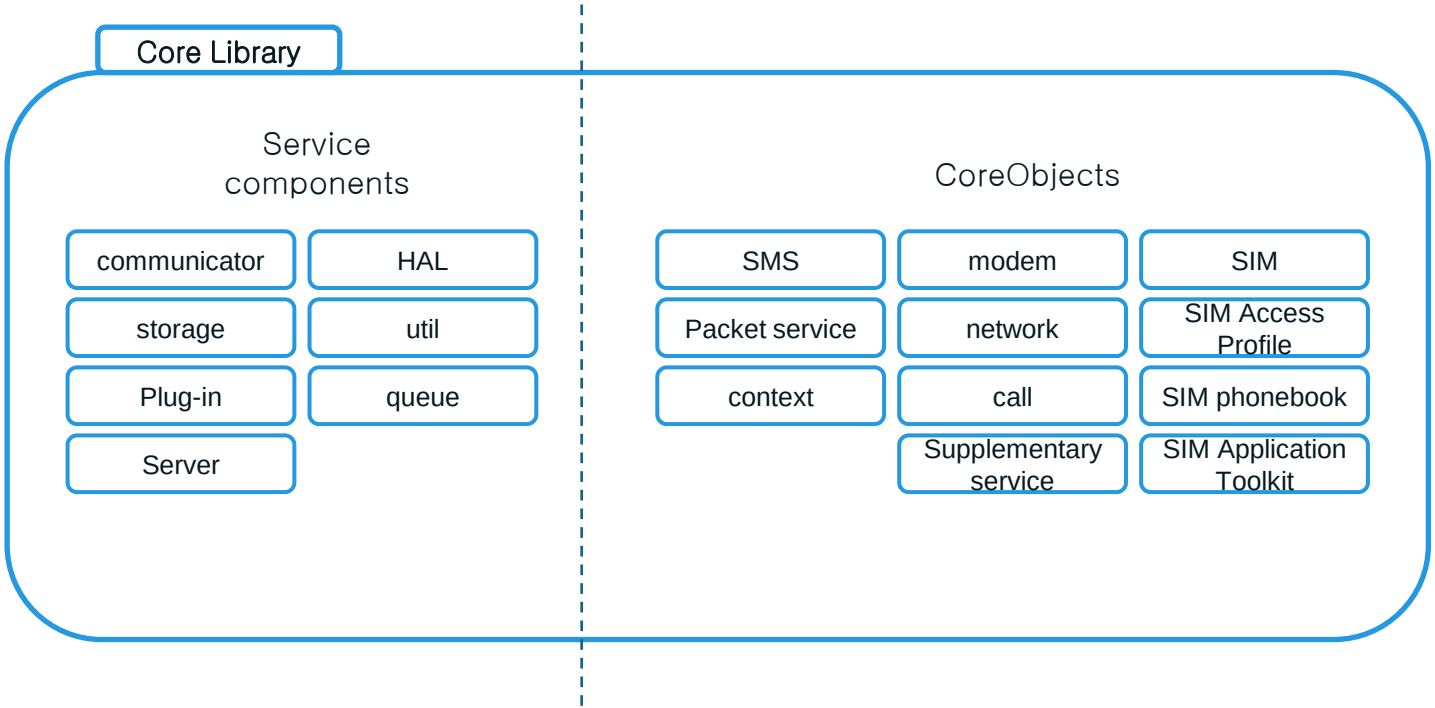


Components

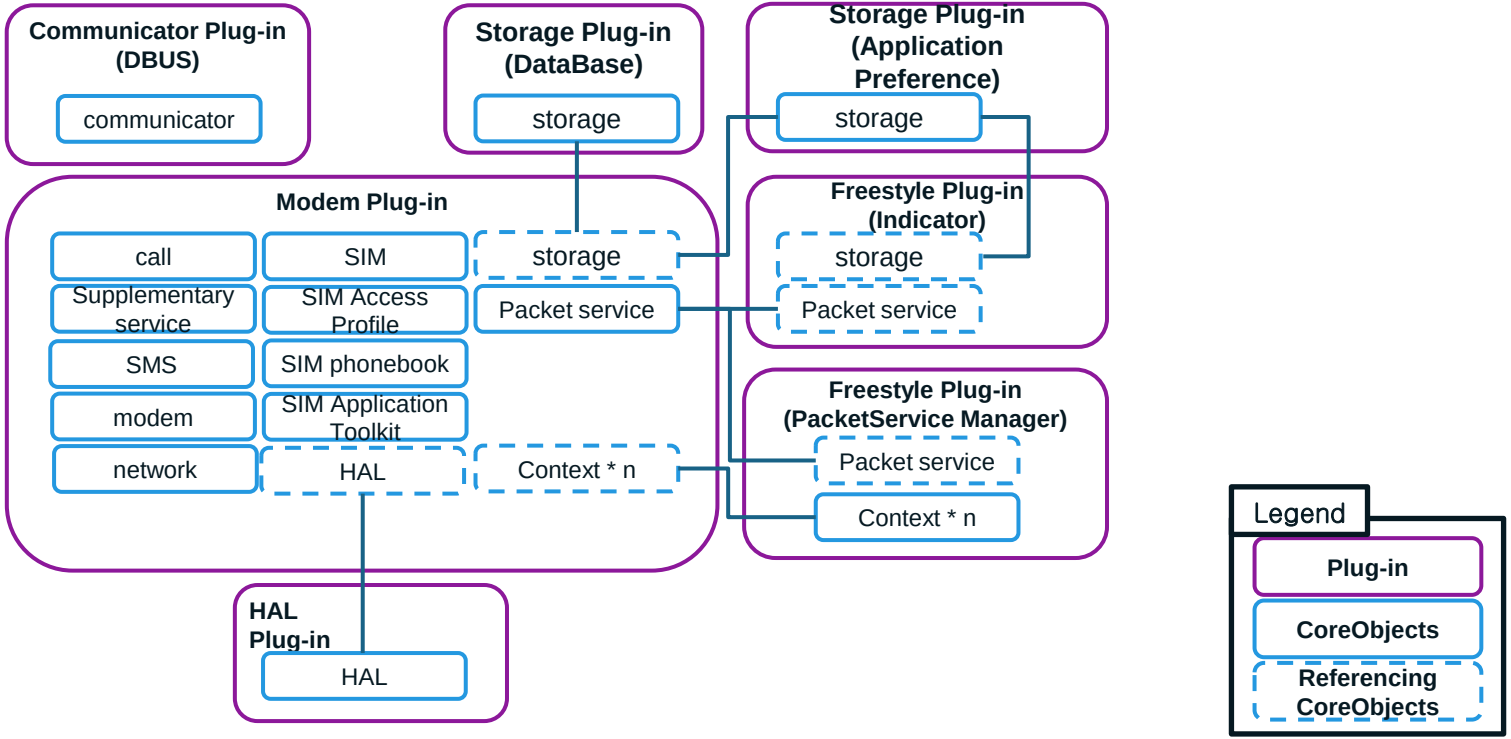
- **Core Library**
  - Base libraries of Tizen Telephony
  - Service Components
    - Server, Plug-in, Queue, HAL, Communicator, Storage, Util
  - Core Objects
    - Functional objects
      - Modem, Network, Call, SS, SMS, PS, Context, SIM, SAP, SAT, SIM Phonebook
    - Operation table
      - Functions of object are defined by operation table
    - Private object
      - Data of objects are stored and can be accessible by get/set APIs
      - e.g : Connected context list
- **Plug-in**
  - Integrated service module
    - Communicator plug-in
      - Interaction between applications and Tizen Telephony stack
    - Modem plug-in
      - Processing requests/responses/notifications between AP and CP
    - Freestyle plug-in
      - Independently processing tasks by a certain trigger
- **Daemon**
  - Dispatcher
    - Sending requests/responses/notifications to a proper plug-in

*AP : Application Processor
*CP : Communication Processor

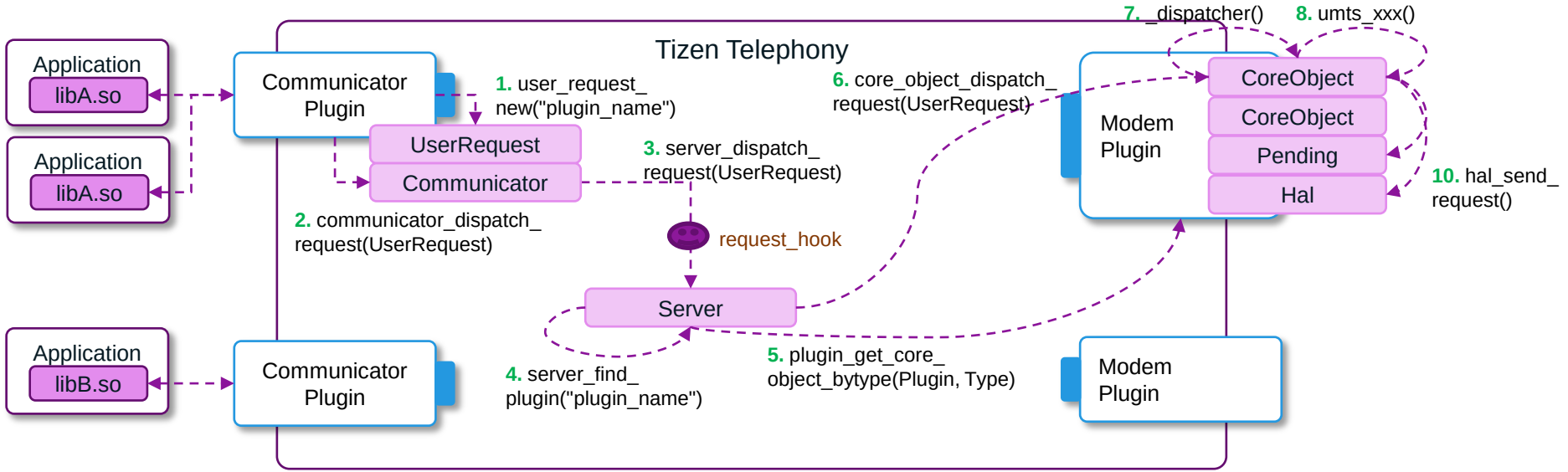
Core Library



Plug-in


Communicator Plug-in (DBUS)
communicator
Storage Plug-in (DataBase)
storage
Storage Plug-in (Application Preference)
storage
Modem Plug-in
call
SIM
storage
Supplementary service
SIM Access Profile
Packet service
SMS
SIM phonebook
modem
SIM Application Toolkit
network
HAL
Context * n
Freestyle Plug-in (Indicator)
storage
Packet service
Freestyle Plug-in (PacketService Manager)
Packet service
Context * n
HAL Plug-in
HAL
Legend
Plug-in
CoreObjects
Referencing CoreObjects

Operation Flow



Developing plug-in

- **Set plug-in description**
  - It should be in any plug-in
  - Symbol (plugin_define_desc) for dynamic loading
  - Defines the name, priority, version, load, init, and unload action

- **Communicator plug-in**
  - Set the operation table
    - Response from modem plug-in
    - Notification from modem plug-in
  - Create the communicator object.

Plug-in description

```
struct tcore_plugin_define_desc
plugin_define_desc =
{
   .name = "MYMODEM",
   .priority =
TCORE_PLUGIN_PRIORITY_MID,
   .version = 1,
   .load = NULL,
   .init = NULL,
   .unload = NULL
};
```

```
struct plugin_define_desc_t  {
   gchar *name;
   enum plugin_priority_e priority;
   int version;
   gboolean (*load)();
   gboolean (*init)(TcorePlugin *);
   void (*unload)(TcorePlugin *);
};
enum plugin_priority_e {
   PLUGIN_PRIORITY_HIGH = -100,
   PLUGIN_PRIORITY_MID = 0,
   PLUGIN_PRIORITY_LOW = +100
};
```

communicator plugin

```
struct communitor_operations_t ops = {
   .send_response = my_send_response,  /* send response to application */
   .send_notification = my_send_notification,  /* send notification to application */
};
On_recv(…)
{
   /* Request delivery to daemon */
   tcore_server_dispatch_request(…)
}

static gboolean on_init(TcorePlugin *p)
{
   Communicator *comm;
   comm = communicator_new(p, &ops);
   …
   /* create socket & bind & listen & accept */
   /* if, recv from application, call on_recv() */
   return TRUE;
}
```

Developing plug-in

- **HAL Plug-in**
  - Create the data channel to modem
  - Naming a certain modem for other plug-ins

HAL plug-in

```
static struct hal_operations_t hops = {
    .power = hal_power,
    .send = hal_send,
};
static gboolean my_hal_recv(GIOChannel *channel, GIOCondition condition, gpointer data)
{
    TcoreHal *hal = data;
     /* read data from fd */
    /* n = length */
    /* buf = read data */
     tcore_hal_emit_recv_callback(hal, n, buf);
}
static gboolean on_init(TcorePlugin *p) {
     TcoreHal *h;

     h = tcore_hal_new(plugin, “myhal”, &hops);

    /* Create MODEM TX/RX Channel */
     fd = … /* create i/o channel for communicate with modem */
     channel = g_io_channel_unix_new(fd);
     source = g_io_add_watch(channel, G_IO_IN, (GIOFunc) my_hal_recv, h);
     g_io_channel_unref(channel);

return TRUE;
}
```

- **Modem Plug-in**
  - Find the HAL for interacting physical modem
  - Initialize the core objects
    - Core objects’ operation table has to be set

modem plug-in

```
static gboolean on_init(TcorePlugin *p)
{
    TcoreHal *h;
     h = tcore_server_find_hal(tcore_plugin_ref_server(p), “myhal”);
    initialize core objects which will be used
    …
    return TRUE;
}
```

Thank You.