0 Outline

1 O-RAN E2 Interface

2 O-RAN E2 Service Models

3 E2 Service Models Architecture in FlexRIC

4 Summary
1 Outline

1 O-RAN E2 Interface

2 O-RAN E2 Service Models

3 E2 Service Models Architecture in FlexRIC

4 Summary
1 RIC Services, E2AP Procedures and E2AP Information Elements

- The E2 interface is composed of RIC Services, E2AP Procedures and E2AP Information Elements (IE)
- There exists 4 RIC Services i.e., Report, Insert, Control and Policy
  - Each Service, is composed of various E2AP Procedures
- There exists 26 E2AP Procedures
  - 10 messages for Near-RT RIC Functional Procedures e.g., RIC Subscription Request, RIC Indication
  - 16 messages for Global Procedures e.g., E2 Setup Request, RIC Service Update
    - Each E2AP, is composed of various E2AP IE
- There exists 32 E2AP IE e.g., Global RIC ID, RIC Indication Header
1 Example: Report RIC Service

Figure: Report Service message sequence chart
1 Example: E2AP RIC Indication Procedure and IE

<table>
<thead>
<tr>
<th>IE/Group Name</th>
<th>Presence</th>
<th>Range</th>
<th>IE type and reference</th>
<th>Semantics description</th>
<th>Criticality</th>
<th>Assigned Criticality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message Type</td>
<td>M</td>
<td></td>
<td>9.2.3</td>
<td></td>
<td>YES</td>
<td>reject</td>
</tr>
<tr>
<td>RIC Request ID</td>
<td>M</td>
<td></td>
<td>9.2.7</td>
<td></td>
<td>YES</td>
<td>reject</td>
</tr>
<tr>
<td>RAN Function ID</td>
<td>M</td>
<td></td>
<td>9.2.8</td>
<td></td>
<td>YES</td>
<td>reject</td>
</tr>
<tr>
<td>RIC Action ID</td>
<td>M</td>
<td></td>
<td>9.2.10</td>
<td></td>
<td>YES</td>
<td>reject</td>
</tr>
<tr>
<td>RIC Indication SN</td>
<td>O</td>
<td></td>
<td>9.2.14</td>
<td></td>
<td>YES</td>
<td>reject</td>
</tr>
<tr>
<td>RIC Indication Type</td>
<td>M</td>
<td></td>
<td>9.2.15</td>
<td></td>
<td>YES</td>
<td>reject</td>
</tr>
<tr>
<td>RIC Indication Header</td>
<td>M</td>
<td></td>
<td>9.2.17</td>
<td></td>
<td>YES</td>
<td>reject</td>
</tr>
<tr>
<td>RIC Indication Message</td>
<td>M</td>
<td></td>
<td>9.2.16</td>
<td></td>
<td>YES</td>
<td>reject</td>
</tr>
<tr>
<td>RIC Call process ID</td>
<td>O</td>
<td></td>
<td>9.2.18</td>
<td></td>
<td>YES</td>
<td>reject</td>
</tr>
</tbody>
</table>

Figure: E2AP RIC Indication Message Procedure definition

<table>
<thead>
<tr>
<th>IE/Group Name</th>
<th>Presence</th>
<th>Range</th>
<th>IE type and reference</th>
<th>Semantics description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RIC Requestor ID</td>
<td>M</td>
<td></td>
<td>INTEGER (0..65535)</td>
<td></td>
</tr>
<tr>
<td>RIC Instance ID</td>
<td>M</td>
<td></td>
<td>INTEGER (0..65535)</td>
<td></td>
</tr>
</tbody>
</table>

Figure: RIC Request ID IE definition
The E2 Interface is composed by Services, Procedures and IE

- Services are composed by many procedures
- Procedures are composed by many Information Elements
- IEs define the type and the valid ranges of the data
# 2 Outline

1. O-RAN E2 Interface

2. O-RAN E2 Service Models

3. E2 Service Models Architecture in FlexRIC

4. Summary
Definition of E2 Service Model: The description of the Services exposed by a specific RAN function within an E2 node over the E2 interface towards the Near-RT RIC

- E2 interface is used to carry messages between a given RAN Function and the Near-RT RIC.
- These messages are RAN Function specific and are described in the corresponding RAN Function specific E2 Service Model.
2 Example: RIC Indication Procedure message

<table>
<thead>
<tr>
<th>IE/Group Name</th>
<th>Presence</th>
<th>Range</th>
<th>IE type and reference</th>
<th>Semantics description</th>
<th>Criticality</th>
<th>Assigned Criticality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message Type</td>
<td>M</td>
<td></td>
<td>9.2.3</td>
<td></td>
<td>YES</td>
<td>reject</td>
</tr>
<tr>
<td>RIC Request ID</td>
<td>M</td>
<td></td>
<td>9.2.7</td>
<td></td>
<td>YES</td>
<td>reject</td>
</tr>
<tr>
<td>RAN Function ID</td>
<td>M</td>
<td></td>
<td>9.2.8</td>
<td></td>
<td>YES</td>
<td>reject</td>
</tr>
<tr>
<td>RIC Action ID</td>
<td>M</td>
<td></td>
<td>9.2.10</td>
<td></td>
<td>YES</td>
<td>reject</td>
</tr>
<tr>
<td>RIC Indication SN</td>
<td>O</td>
<td></td>
<td>9.2.14</td>
<td></td>
<td>YES</td>
<td>reject</td>
</tr>
<tr>
<td>RIC Indication Type</td>
<td>M</td>
<td></td>
<td>9.2.15</td>
<td></td>
<td>YES</td>
<td>reject</td>
</tr>
<tr>
<td>RIC Indication Header</td>
<td>M</td>
<td></td>
<td>9.2.17</td>
<td></td>
<td>YES</td>
<td>reject</td>
</tr>
<tr>
<td>RIC Indication Message</td>
<td>M</td>
<td></td>
<td>9.2.16</td>
<td></td>
<td>YES</td>
<td>reject</td>
</tr>
<tr>
<td>RIC Call process ID</td>
<td>O</td>
<td></td>
<td>9.2.18</td>
<td></td>
<td>YES</td>
<td>reject</td>
</tr>
</tbody>
</table>

Figure: E2AP RIC Indication Message Procedure definition

<table>
<thead>
<tr>
<th>IE/Group Name</th>
<th>Presence</th>
<th>Range</th>
<th>IE type and reference</th>
<th>Semantics description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RIC Indication message</td>
<td>M</td>
<td></td>
<td>OCTET STRING</td>
<td>Defined in RAN Function specific E2 Service Model [3]</td>
</tr>
</tbody>
</table>

Figure: RIC Indication Message IE definition
There are 5 E2AP Procedures that a SM should provide
There are 9 IE that a E2SM should provide

<table>
<thead>
<tr>
<th>RAN Function specific E2AP Information Elements</th>
<th>E2AP Information Element reference</th>
<th>Related E2AP Procedures</th>
</tr>
</thead>
<tbody>
<tr>
<td>RIC Event Trigger Definition IE</td>
<td>E2AP [3] section 9.2.9</td>
<td>RIC Subscription</td>
</tr>
<tr>
<td>RIC Action Definition IE</td>
<td>E2AP [3] section 9.2.12</td>
<td>RIC Subscription</td>
</tr>
<tr>
<td>RIC Indication Message IE</td>
<td>E2AP [3] section 9.2.16</td>
<td>RIC Indication</td>
</tr>
<tr>
<td>RIC Call Process ID IE</td>
<td>E2AP [3] section 9.2.18</td>
<td>RIC Indication, RIC Control</td>
</tr>
<tr>
<td>RIC Control Header IE</td>
<td>E2AP [3] section 9.2.20</td>
<td>RIC Control</td>
</tr>
<tr>
<td>RIC Control Message IE</td>
<td>E2AP [3] section 9.2.19</td>
<td>RIC Control</td>
</tr>
<tr>
<td>RIC Control Outcome IE</td>
<td>E2AP [3] section 9.2.25</td>
<td>RIC Control</td>
</tr>
</tbody>
</table>

Figure: Relationship between E2AP Procedures and IE of a SM
SM exposes the information of a specific RAN function over the E2 interface towards the Near-RT RIC

E2SM information is embedded within a E2AP Procedure as raw bytes

Every SM provides the semantic to interpret the data of 9 IE that are transported within 5 procedures

E2 interface enables decoupling the protocol i.e., E2AP from the data, which is interpreted by the E2SMs
3 Outline

1 O-RAN E2 Interface

2 O-RAN E2 Service Models

3 E2 Service Models Architecture in FlexRIC

4 Summary
3 FlexRIC SM as plug-ins

- SM are designed as plug-ins (shared dynamic libraries)
- SMs need to be loaded in the Near-RT RIC, as well as in the E2 Agent
- SM are built through the Factory Method Pattern, and thus, their procedures are their interface

```c
#ifndef PDCP_SERVICE_MODEL_AGENT_H
#define PDCP_SERVICE_MODEL_AGENT_H

#include <stdio.h>
#include <stdint.h>

#include "./sm_agent.h"

sm_agent_t* make_pdcp_sm_agent(sm_io_ag_t io);

#endif
```

Figure: PDCP SM Factory Pattern function
SMs communicate with the RAN through a read and write functions that is provided in construction, which facilitates its portability.

```c
const char server_ip_str[] = "127.0.0.1";
const gNB_RRC_INST* rrc = RC.nrrrc[mod_id];
assert(rrc);

const int mcc = rrc->configuration.mcc[0]; // 208;
const int mnc = rrc->configuration.mnc[0]; // 94;
const int mnc_digit_len = rrc->configuration.mnc_digit_length[0]; // 2;
const int nb_id = rrc->configuration.cell_identity; // 42;
sm_io_ag_t io = {.read = read_RAN, .write = write_RAN};

init_agent_api(server_ip_str, mcc, mnc, mnc_digit_len, nb_id, io);
```

Figure: FlexRIC E2 Agent initialization in OAI and CMakeLists.txt diff
SMs provide the 5 procedures and the 9 IEs specified by O-RAN

```c
typedef struct {
    sm_subs_data_t (*on_subscription)(sm_ric_t const*, const char* cmd);
    sm_rd_if_t (*on_indication)(sm_ric_t const*, sm_ind_data_t* data);
    sm_ctrl_data_t (*on_control)(sm_ric_t const*, const char*);
    void (*on_e2_setup)(sm_ric_t const*, const sm_e2_setup_t*);
    sm_ric_service_update_t (*on_ric_service_update)(sm_ric_t const*, const char*);
} sm_e2ap_procedures_ric_t;

typedef struct sm_ric_s {

```

Figure: 5 Procedures defined in the SM for the RIC
The SMs offer a C based struct definition, that it is used as data IR

```c
typedef struct{
    uint32_t txpdu_pkts;  /* aggregated number of tx packets */
    uint32_t txpdu_bytes; /* aggregated bytes of tx packets */
    uint32_t txpdu_sn;    /* current sequence number of last tx packet (or TX_NEXT) */
    uint32_t rxpdu_pkts;  /* aggregated number of rx packets */
    uint32_t rxpdu_bytes; /* aggregated bytes of rx packets */
    uint32_t rxpdu_sn;    /* current sequence number of last rx packet (or RX_NEXT) */
    uint32_t rxpdu ОО_pkts; /* aggregated number of out-of-order rx pkts (or RX_REORD) */
    uint32_t rxpdu ОО_bytes; /* aggregated amount of out-of-order rx bytes */
    uint32_t rxpdu dd_pkts; /* aggregated number of duplicated discarded packets */
    uint32_t rxpdu dd_bytes; /* aggregated amount of discarded packets' bytes */
    uint32_t rxpdu ro_count; /* this state variable indicates the COUNT value */
    uint32_t txsd_reports; /* number of SDUs delivered */
    uint32_t txsd_bytes;  /* number of bytes of SDUs delivered */
    uint32_t rxsd_reports; /* number of SDUs received */
    uint32_t rxsd_bytes;  /* number of bytes of SDUs received */
    uint32_t rnti;
    uint8_t mode;        /* 0: PDCP AM, 1: PDCP UM, 2: PDCP TM */
    uint8_t rbid;
} pdcp_radio_bearer_stats_t;

typedef struct{
    uint32_t len;
    pdcp_radio_bearer_stats_t* rb;
    uint16_t frame;
    uint8_t slot;
} pdcp_ind_msg_t;
```

Figure: One of the nine IE that a SM defines
IE encoding and decoding is specified at compile time through C11 __Generic e.g., plain, ASN.1, Flatbuffers i.e., zero runtime cost overhead while offering great flexibility.

```c
#define pdcp_enc_ind_hdr(T,U) __Generic ((T), \ 
    pdcp_enc_plain_t*: pdcp_enc_ind_hdr_plain , \ 
    pdcp_enc_asn_t*: pdcp_enc_ind_hdr_asn, \ 
    pdcp_enc_fb_t*: pdcp_enc_ind_hdr_fb, \ 
    default: pdcp_enc_ind_hdr_plain) (U)

#define pdcp_enc_ind_msg(T,U) __Generic ((T), \ 
    pdcp_enc_plain_t*: pdcp_enc_ind_msg_plain , \ 
    pdcp_enc_asn_t*: pdcp_enc_ind_msg_asn, \ 
    pdcp_enc_fb_t*: pdcp_enc_ind_msg_fb, \ 
    default: pdcp_enc_ind_msg_plain) (U)
```

Figure: Encoding static polymorphism
3 Example of Indication msg: FlexRIC blocks

- ASIO: Asynchronous IO e.g., epoll
- EP: Endpoint e.g., SCTP connection
- AP: Application protocol e.g., encoding/decoding in ASN.1 or Flatbuffers
- MSG Handler e.g., event based logical block
- SM: Service Models i.e., loaded SMs
- iApps: Internal Apps e.g., write to a DB

Figure: FlexRIC agent and RIC blocks
3 Example of Indication msg: Indication event on the agent

- Event is generated e.g., timeout
- The message handler calls the corresponding SM function e.g., on_indication
- The SM communicates with the RAN, fills the specific IE, encodes it and returns a byte array
- The message handler composes the e2ap_msg_t and calls the AP
- The AP encodes the e2ap_msg_t e.g., ASN.1 or FB
- The bytes are sent through the endpoint

Figure: FlexRIC agent
3 Example of Indication msg: Indication event on the RIC

- A new event happened i.e., a new packet arrived
- The AP decodes the e2ap_msg_t
- The message handler calls the corresponding SM
- The message handler forwards the data from the SM to the subscribed iApps

Figure: FlexRIC RIC
3 Example of Indication msg: Indication event Agent

```c
} else if(e.type == INDICATION_EVENT){

    sm_agent_t* sm = e.i_ev->sm;
    sm_ind_data_t data = sm->proc.on_indication(sm);

    ric_indication_t ind = generate_indication(ag, &data, e.i_ev);
    defer({ e2ap_free_indication(&ind); })
;

    byte_array_t ba = e2ap_enc_indication_ag(&ag->ap, &ind);
    defer({ free_byte_array(ba); })
;
    e2ap_send_bytes_agent(&ag->ep, ba);

```
3 Example of Indication msg: Indication event RIC

```c
if(net_pkt(ric, fd) == true){
    byte_array_t ba = e2ap_recv_msg_ric(&ric->ep);
    defer({free_byte_array(ba); });

    e2ap_msg_t msg = e2ap_msg_dec_ric(&ric->ap, ba);
    defer({e2ap_msg_free_ric(&ric->ap, &msg); });

    e2ap_msg_t ans = e2ap_msg_handle_ric(ric, &msg);
    defer({e2ap_msg_free_ric(&ric->ap, &ans);});

    if(ans.type != NONE_E2_MSG_TYPE ){
        byte_array_t ba_ans = e2ap_msg_enc_ric(&ric->ap, &ans);
        defer({free_byte_array(ba_ans); });
        e2ap_send_bytes_ric(&ric->ep, ba_ans);
    }
}
```

![Diagram](image-url)
3 Example of Indication msg: Handle Indication message RIC

```c
// E2 -> RIC
e2ap_msg_t e2ap_handle_indication_ric(near_ric_t* ric, const e2ap_msg_t* msg)
{
    assert(ric != NULL);
    assert(msg != NULL);
    assert(msg->type == RIC_INDICATION);

    ric_indication_t const* ric_ind = &msg->umsgs.ric_ind;

    const uint16_t ran_func_id = ric_ind->ric_id.ran_func_id;
    sm_ric_t* sm = sm_plugin_ric(&ric->plugin, ran_func_id);

    sm_ind_data_t data = {.ind_hdr = ric_ind->hdr.buf,
                          .len_hdr = ric_ind->hdr.len,
                          .ind_msg = ric_ind->msg.buf,
                          .len_msg = ric_ind->msg.len,
                        };

    if(ric_ind->call_process_id != NULL){
        data.call_process_id = ric_ind->call_process_id->buf;
        data.len_cpid = ric_ind->call_process_id->len;
    }

    sm_rd_if_t d = sm->proc.on_indication(sm, &data);
    defer({sm->alloc.free_ind_msg(&d); });
    assert(d.type == MAC_STATS_V0 || d.type == RLC_STATS_V0 || d.type == PDCP_STATS_V0);

    publish_ind_msg(ric, ran_func_id, &d);

    e2ap_msg_t ans = {.type = NONE_E2_MSG_TYPE };
    return ans;
}
```
4 Outline

1. O-RAN E2 Interface
2. O-RAN E2 Service Models
3. E2 Service Models Architecture in FlexRIC
4. Summary
4 Summary

- FlexRIC is O-RAN E2 standards compliant and closely follows the standard specifications
- FlexRIC achieves flexibility and extensability through the SM plugin system
- A SM needs to implement 5 procedures and 9 IEs for the agent and the RIC
- SM are not coupled to a encoding and decoding scheme. It's static polymorphism permits changing the scheme smoothly
Thank you!