Linux CAIF

Author: Sjur Brendeland/ sjur.brandeland@stericsson.com

Introduction

CAIF is a MUX protocol used by ST-Ericsson cellular modems for communication between Modem and host. The host processes can open virtual AT channels, initiate GPRS Data connections, Video channels and Utility Channels. The Utility Channels are general purpose pipes between modem and host.

ST-Ericsson modems support a number of transports between modem and host. Currently, UART and Loopback are available for Linux.

Architecture:

The implementation of CAIF is divided into:
* CAIF Socket Layer, Kernel API, and Net Device.
* CAIF Core Protocol Implementation
* CAIF Link Layer, implemented as NET devices.

```
  RTNL
   !
   !     +------+   +------+   +------+
   !    +------+!  +------+!  +------+!
   !    ! Sock !!  !Kernel!!  ! Net  !!
   !    ! API  !+  ! API  !+  ! Dev  !+   <- CAIF Client APIs
   !    +------+   +------!   +------+
   !       !          !          !
   !       +----------!----------+
   !               +------+               <- CAIF Protocol Implementation
   +------->       ! CAIF !
                   ! Core !
                   +------+
             +--------!--------+
             !                 !
          +------+          +-----+
          !      !          ! TTY !       <- Link Layer (Net Devices)
          +------+          +-----+
```

Using the Kernel API

The Kernel API is used for accessing CAIF channels from the kernel.
The user of the API has to implement two callbacks for receive and control.
The receive callback gives a CAIF packet as a SKB. The control callback will notify of channel initialization complete, and flow-on/flow-off.

```
  struct caif_device caif_dev = {
    .caif_config = {
     .name = "MYDEV"
     .type = CAIF_CHTY_AT
    }
   .receive_cb = my_receive,
   .control_cb = my_control,
  };
  caif_add_device(&caif_dev);
  caif_transmit(&caif_dev, skb);
```

See the caif_kernel.h for details about the CAIF kernel API.

IMPLEMENTATION

===========================
===========================

CAIF Core Protocol Layer

===

CAIF Core layer implements the CAIF protocol as defined by ST-Ericsson. It implements the CAIF protocol stack in a layered approach, where each layer described in the specification is implemented as a separate layer. The architecture is inspired by the design patterns "Protocol Layer" and "Protocol Packet".

== CAIF structure ==
The Core CAIF implementation contains:

- Simple implementation of CAIF.
- Layered architecture (a la Streams), each layer in the CAIF specification is implemented in a separate c-file.
- Clients must implement PHY layer to access physical HW with receive and transmit functions.
- Clients must call configuration function to add PHY layer.
- Clients must implement CAIF layer to consume/produce CAIF payload with receive and transmit functions.
- Clients must call configuration function to add and connect the Client layer.
- When receiving / transmitting CAIF Packets (cfpkt), ownership is passed to the called function (except for framing layers' receive functions or if a transmit function returns an error, in which case the caller must free the packet).

Layered Architecture

The CAIF protocol can be divided into two parts: Support functions and Protocol Implementation. The support functions include:

- CFPKT CAIF Packet. Implementation of CAIF Protocol Packet. The CAIF Packet has functions for creating, destroying and adding content and for adding/extracting header and trailers to protocol packets.

- CFLST CAIF list implementation.

- CFGLUE CAIF Glue. Contains OS Specifics, such as memory allocation, endianness, etc.

The CAIF Protocol implementation contains:

- CFCNFG CAIF Configuration layer. Configures the CAIF Protocol Stack and provides a Client interface for adding Link-Layer and Driver interfaces on top of the CAIF Stack.

- CFCTRL CAIF Control layer. Encodes and Decodes control messages such as enumeration and channel setup. Also matches request and response messages.

- CFSERVL General CAIF Service Layer functionality; handles flow control and remote shutdown requests.

- CFVEI CAIF VEI layer. Handles CAIF AT Channels on VEI (Virtual External Interface). This layer encodes/decodes VEI frames.

- CFDGML CAIF Datagram layer. Handles CAIF Datagram layer (IP traffic), encodes/decodes Datagram frames.

- CFMUX CAIF Mux layer. Handles multiplexing between multiple physical bearers and multiple channels such as VEI, Datagram, etc. The MUX keeps track of the existing CAIF Channels and Physical Instances and selects the apropriate instance based on Channel-Id and Physical-ID.

- CFFRML CAIF Framing layer. Handles Framing i.e. Frame length and frame checksum.

- CFSERL CAIF Serial layer. Handles concatenation/split of frames into CAIF Frames with correct length.

```
                    +----------+
                    | Config   |
                    | CFCNFG   |
                    +----------+
                         !
    +----------+    +----------+    +----------+
    |    AT    |    | Control  |    | Datagram |
    | CFVEIL   |    | CFCTRL   |    | CFDGML   |
    +----------+    +----------+    +----------+
           \_____________!______________/
                         !
                    +----------+
                    |   MUX    |
                    |          |
                    +----------+
                    _____!_____
                   /           \
            +----------+    +----------+
            | CFFRML   |    | CFFRML   |
            | Framing |      | Framing |
            +----------+     +----------+
                 !                !
            +----------+     +----------+
            |          |     | Serial   |
            |          |     | CFSERL   |
            +----------+     +----------+
```

In this layered approach the following "rules" apply.

- All layers embed the same structure "struct cflayer"
- A layer does not depend on any other layer's private data.
- Layers are stacked by setting the pointers
  `layer->up , layer->dn`
- In order to send data upwards, each layer should do
  `layer->up->receive(layer->up, packet);`
- In order to send data downwards, each layer should do
  `layer->dn->transmit(layer->dn, packet);`

Linux Driver Implementation

Linux GPRS Net Device and CAIF socket are implemented on top of the CAIF Core protocol. The Net device and CAIF socket have an instance of 'struct cflayer', just like the CAIF Core protocol stack. Net device and Socket implement the 'receive()' function defined by 'struct cflayer', just like the rest of the CAIF stack. In this way, transmit and receive of packets is handled as by the rest of the layers: the 'dn->transmit()' function is called in order to transmit data.

The layer on top of the CAIF Core implementation is sometimes referred to as the "Client layer".

Configuration of Link Layer

The Link Layer is implemented as Linux net devices (struct net_device). Payload handling and registration is done using standard Linux mechanisms.

The CAIF Protocol relies on a loss-less link layer without implementing retransmission. This implies that packet drops must not happen. Therefore a flow-control mechanism is implemented where the physical interface can initiate flow stop for all CAIF Channels.