LINUX PLUMBERS CONFERENCE September 20-24, 2021

Apps not boilerplate, leveraging Android's CHRE and Zephyr

Yuval Peress

Motivation

- Reduce time to market
- Reduce cost of maintenance and updates
- Reduce implementation complexity
- Improve testability
- Improve modularity and code reusability

Developers should be able to focus on WHAT they're building, not HOW.

Motivation by pseudo-code (timestamp spreading only)

```
static void do_something_interesting(sample) {
  // This is our actual code
}

static void read_accelerometer(...) {
  while (has_samples(accel)) {
    sample = read_sample(accel);
    calculate_timestamp(sample);
    post_to_event_loop(sample);
  }
}

static void interrupt_handler(...) {
  save_current_timestamp();
  post_handler(read_accelerometer);
}
```

```
void event_loop_thread() {
  while (1) {
    sample = poll_samples();
    if (sample) {
      broadcast_sample(sample);
    }
    yield();
  }
}

int main() {
  register_interrupt_handler(accel,
                             interrupt_handler);
  create_event_loop_thread();
  subscribe_to_events(do_something_interesting);

  ...
}
```

Motivation by pseudo-code (timestamp spreading only)

```
static void nanoappStart(void) {
  request_events(TYPE_ACCEL, …);
}

static void nanoappHandleEvent(event) {
  // This is our actual code
}
```

What is Zephyr?

- An embedded RTOS that is currently being integrated into chromium's EC.

What is Zephyr?

- An embedded RTOS that is currently being integrated into chromium's EC.
- Uses devicetree to specify how the board is connected.

```
&i2c0 {
  /* Add BMI160 to I2C bus */
  accel: bmi@68 {
    compatible = "bosch,bmi160";
    reg = <0x68>;
    label = "accel-i2c";
  };
};
```

Zephyr™

What is Zephyr?

- An embedded RTOS that is currently being integrated into chromium's EC.
- Uses devicetree to specify how the board is connected.
- Provides common device APIs to abstract hardware details.

```
struct sensor_value val;
const struct device *dev =
    device_get_binding(
        DT_LABEL(DT_NODELABEL(accel)));

/* Read the X value from an accelerometer. */
sensor_channel_get(dev, SENSOR_CHAN_ACCEL_X, &val);
```

Zephyr™

What is CHRE?

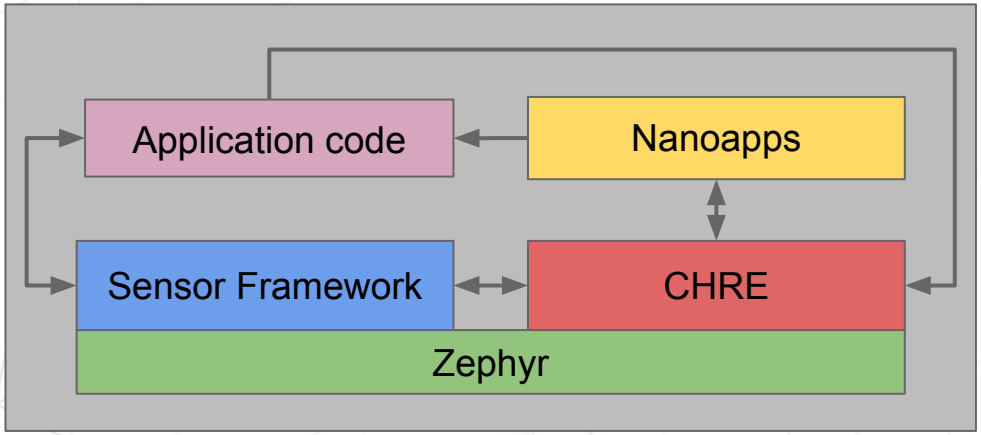
- Context Hub Runtime Environment
- Provides a framework for running nanoapps
  - Small *feature* applications that run on the EC and generally provide some functionality to the application or another processor by running in a low power environment. Examples: lid angle calculation, online sensor calibration, geofencing, WiFi scanning, and more.
  - Have 3 entry points
    - `nanoappStart()`
    - `nanoappStop()`
    - `nanoappHandleEvent()`
- Manages events in a pub/sub like model between nanoapps and peripheral frameworks

Advanced CHRE features

- *Supports limited memory footprint*:
  able to dynamically load/unload/start/stop nanoapps.
- *Low risk updates*:
  an OTA update can just update nanoapps
- *Supports a large array of peripherals*:
  WiFi, BT, GNSS, IMU sensors, audio, WWAN.

How it all fits together

- Everything uses Zephyr at the core.
- *Sensor Framework* uses devicetree to configure itself and will communicate directly with the CHRE as well as with the application using a custom *TX Layer*.
- Nanoapps can be added statically or dynamically and may communicate with the CHRE or with the application using the same *TX layer* as the frameworks..
- Other frameworks (WiFi, GNSS, etc) can also be added in the future.



The Sensor Framework/Subsystem

- Timestamp spreading
- Sample rate arbitration
- Sample batching
- Support automated power management modes

Why should I care?

Reduce time to market

To get an app going you just need the devicetree files and the nanoapp that consumes the events.

Reduce cost

Maintenance and updates to the RTOS, drivers, event routing, and frameworks are community responsibilities.

Reduce implementation complexity

The modularity of the components means that developers can focus on one thing at a time. The problem is no longer a system design problem, but building a product.

Improve testability, modularity, and reusability

Since each nanoapp has well defined input and output events, the system as a whole is much more modular. Comprised of components that are easily tested and reused.

How far can this go?

- Bicycle automatic transmission? Nanoapp that consumes torque, power, and cadence from and shifts gears.
- Wearable activity detection and tracking? Nanoapps for swimming, running, cycling, etc.
- Smart scale? Nanoapp to compute body metrics from impedance.
- *NPM* like nanoapp package manager?

If Zephyr was chosen to offload the RTOS components of the EC, CHRE can be thought of as offloading the framework.

Further readings:

- Zephyr - https://www.zephyrproject.org/
- CHRE - https://source.android.com/devices/contexthub

Discussion...