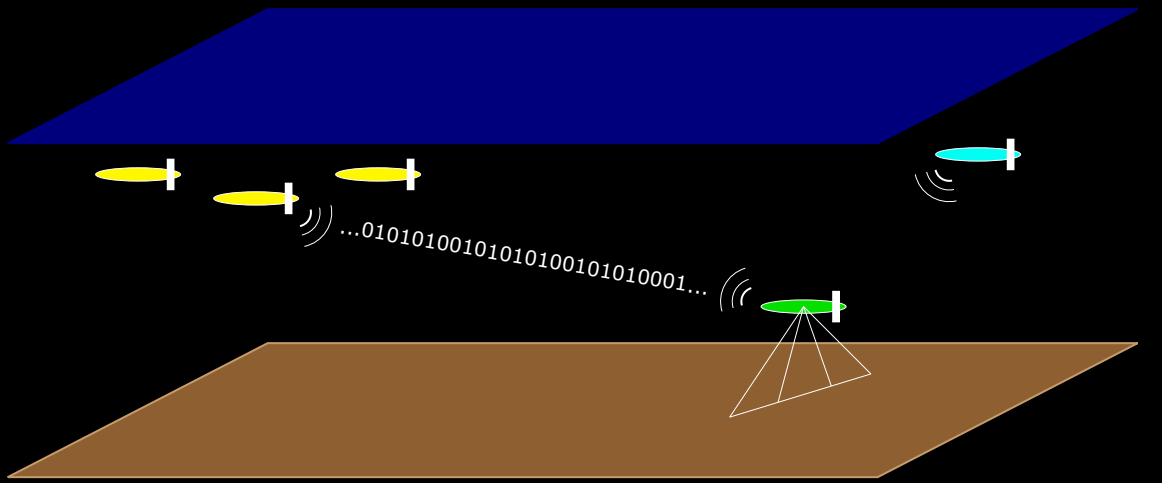
NRL Goby3 Course

Lecture 2: Using Goby3 MOOS Applications



Toby Schneider

GobySoft, LLC

Mashpee, MA, USA

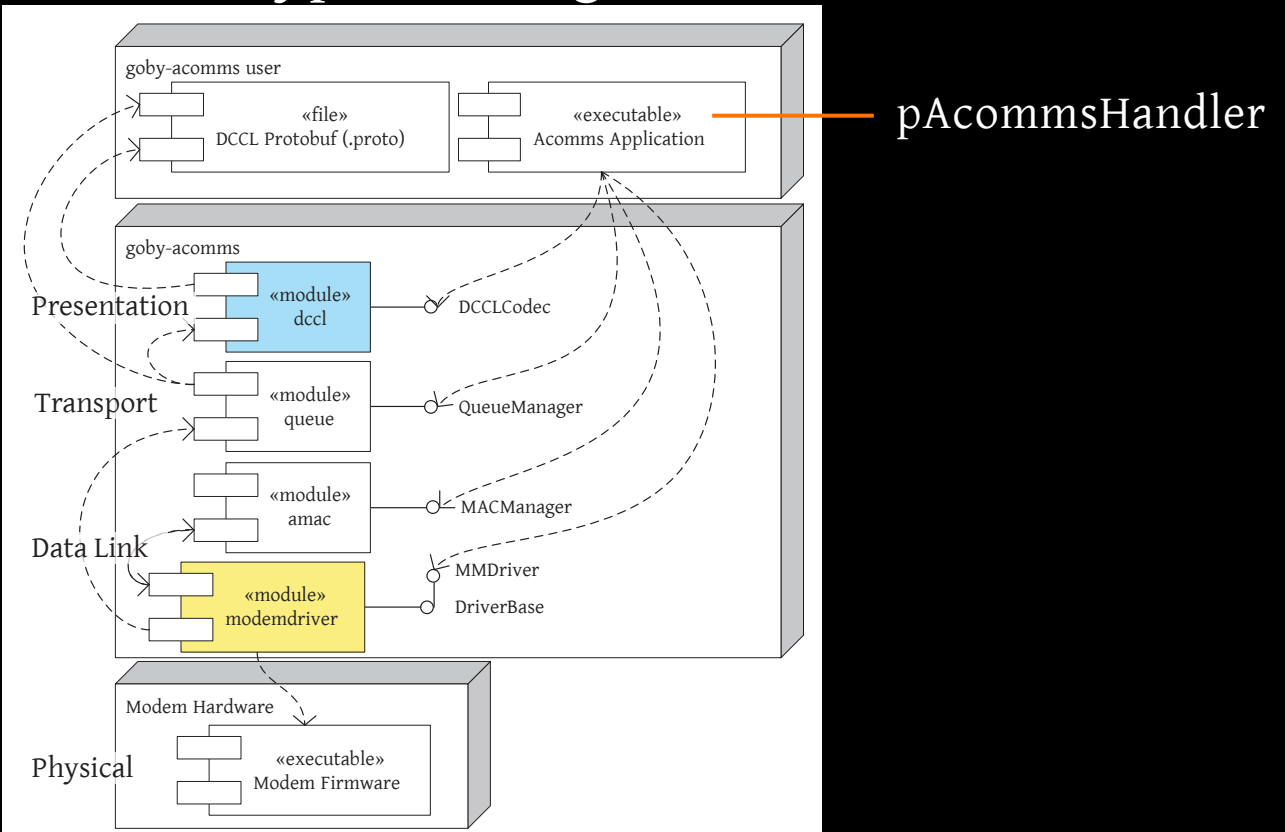
gobysoft.org
aquatic software

pAcommsHandler (Goby-Acomms in MOOS)

pAcommsHandler provides a:

1. MOOS Application wrapper for the Goby-Acomms communication library.

2. set of translation tools for converting the DCCL messages to MOOS types (strings and doubles) and vice-versa



pAcommsHandler Configuration

```
ProcessConfig = pAcommsHandler
{
  common {  # Configuration common to all Goby MOOS applications
    ...
  }
  modem_id: 1  # Unique number 1-31 to identify this node

  driver_cfg {  # Configure the primary acoustic modem driver
                # (used for sending and receiving)
    ...
  }
  mac_cfg {  # Configure the acoustic Medium Access Control
    ...
  }
  queue_cfg {  # Configure the Priority Queuing layer
    ...
  }
  moos_var {  # MOOS Variables published by pAcommsHandler
    ...
  }
  load_shared_library: "/usr/lib/libmy_dccl_messages.so"
  translator_entry {  # Translate between MOOS and DCCL
    ...
  }
}
```

pAcommsHandler Conf: Common

```
ProcessConfig = pAcommsHandler
{
    common {
        verbosity: DEBUG1 # QUIET, WARN, VERBOSE, DEBUG1, DEBUG2,
                          # DEBUG3
        show_gui: true
    }
    ...
}
```

"pAcommsHandler -e" will always give ALL valid configuration variables

pAcommsHandler Conf: Queue

```
queue_cfg
 {
     message_entry {
         protobuf_name: "goby3_course.dccl.NavigationReport"
         ack: false          # (default=true)
         blackout_time: 0    # (default=0) (seconds)
         max_queue: 10       # (default=100) (count)
         newest_first: true  # (default=true)
         ttl: 1800           # (default=1800) (seconds)
         value_base: 1       #  (default=1)
     }
 }
```

pAcommsHandler Conf: AMAC

```
mac_cfg {
    type: MAC_FIXED_DECENTRALIZED
    slot { src: 1 slot_seconds: 10 max_frame_bytes: 128 }
    slot { src: 3 slot_seconds: 10 max_frame_bytes: 128 }
 }
```

Just a list of ModemTransmission messages. Some drivers require max_frame_size to be specified, some use the rate to determine this (MicroModem driver).

pAcommsHandler Conf: Driver

```
driver_cfg {
      driver_type: DRIVER_UDP_MULTICAST
      [goby.acomms.udp_multicast.protobuf.config] {
          multicast_address: “239.142.0.2”
          multicast_port: 54500
          max_frame_size: 1400
      }
  }
```

```
driver_cfg {
      driver_type: DRIVER_WHOI_MICROMODEM
      serial_port: “/tmp/ttymm0”
      serial_baud: 19200
      [goby.acomms.micromodem.protobuf.config] {
         reset_nvram: true
      }
 }
```

pAcommsHandler Conf: Translator

```
load_shared_library: "../../build/lib/libgoby3_course_messages.so"
 translator_entry {
     protobuf_name: "goby3_course.dccl.NavigationReport"
     trigger {
         type: TRIGGER_PUBLISH
         moos_var: "NAVIGATION_REPORT_OUT"
     }
     create {
         technique: TECHNIQUE_PREFIXED_PROTOBUF_TEXT_FORMAT
         moos_var: "NAVIGATION_REPORT_OUT"
     }
     publish {
         technique: TECHNIQUE_PREFIXED_PROTOBUF_TEXT_FORMAT
         moos_var: "NAVIGATION_REPORT_IN"
     }
 }
```

pAcommsHandler Conf: Translator

Selected Translator Entry techniques:

- TECHNIQUE_PREFIXED_PROTOBUF_TEXT_FORMAT
  - google::protobuf::TextFormat parser/serializer with a prefix of "@PB[MessageName]"
  - Widely used in Goby-MOOS because it is compatible with existing tools (e.g. uXMS)
- TECHNIQUE_FORMAT (useful for legacy conversions)
  - Like printf or scanf, but where type specifiers are replaced with numeric field specifiers where the number is the protobuf field tag ID:
    %d (printf/scanf) --> %N% (Goby)
- TECHNIQUE_PROTOBUF_NATIVE_ENCODED
  - Uses the Protocol Buffers binary encoding. Fast and efficient, but not compatible with MOOS tools that use human-readable strings.

pAcommsHandler MOOS interface

Driver related:

- ACOMMS_RAW_INCOMING / ACOMMS_RAW_OUTGOING (data to/from modem)
- ACOMMS_MODEM_RECEIVE (received data from modem driver)
- ACOMMS_MODEM_TRANSMIT (transmit data from modem driver)

pAcommsHandler MOOS interface

Queue related:

- ACOMMS_QUEUE_RECEIVE (received data from Queue)
- ACOMMS_QUEUE_TRANSMIT (transmit data from Queue)
- ACOMMS_ACK: Acknowledgment message
- ACOMMS_ACK_ORIGINAL: Copy of the message that was just acknowledged
- ACOMMS_EXPIRE: Queue expiration message
- ACOMMS_QSIZE: Queue size message (sent when the queue grows or shrinks
- ACOMMS_FLUSH_QUEUE: Command to empty (flush) a queue

pAcommsHandler MOOS interface

MAC related:

- ACOMMS_MAC_CYCLE_UPDATE: Online MAC cycle update
- ACOMMS_MAC_INITIATE_TRANSMISSION: Start of a MAC cycle when this modem is transmitting
- ACOMMS_MAC_SLOT_START: Sent at the start of any MAC slot (including one that this modem is not transmitting on)

pAcommsHandler MOOS interface

DCCL related:

- Completely configured in the TranslatorEntry configuration!
  - create and trigger: subscribe
  - publish: publish

pTranslator

pTranslator is simply pAcommsHandler without any of the acomms.

- Just translator_entry { } blocks
- Useful for systems that need MOOS <-> Protobuf translation but not acomms.

"Frontseat" - "backseat" abstraction

Useful abstraction:

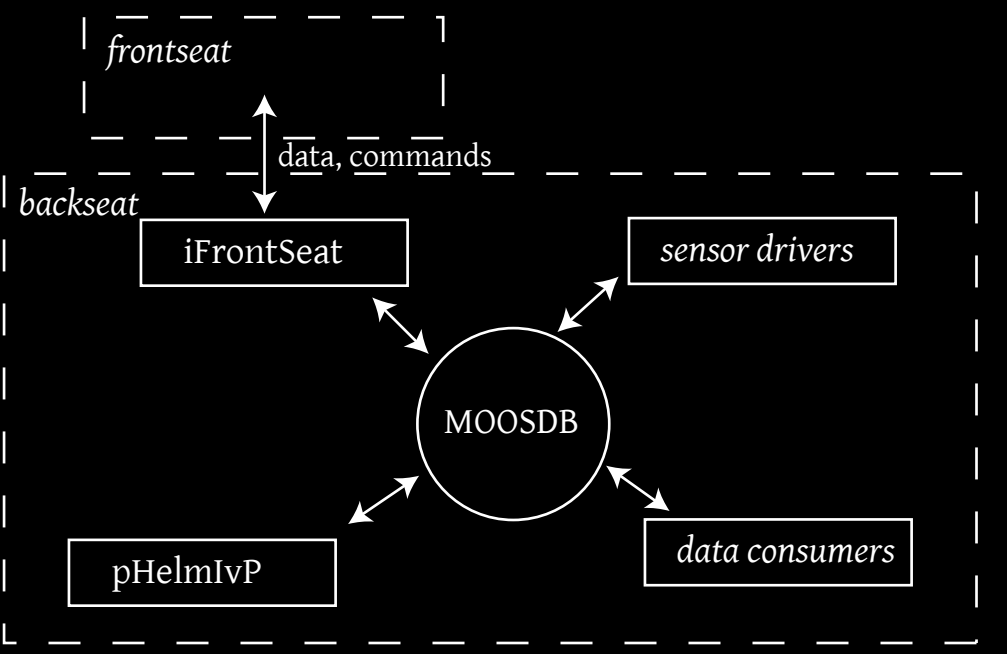
- AUV / USV manufacturer computer = "frontseat"
- MOOS-IvP payload computer = "backseat"

Allows for LAMSS autonomy suite to be deployed on a wide variety of different vehicles with different manufacturers.

What is a “frontseat interface”?

The frontseat interface has several main functions:

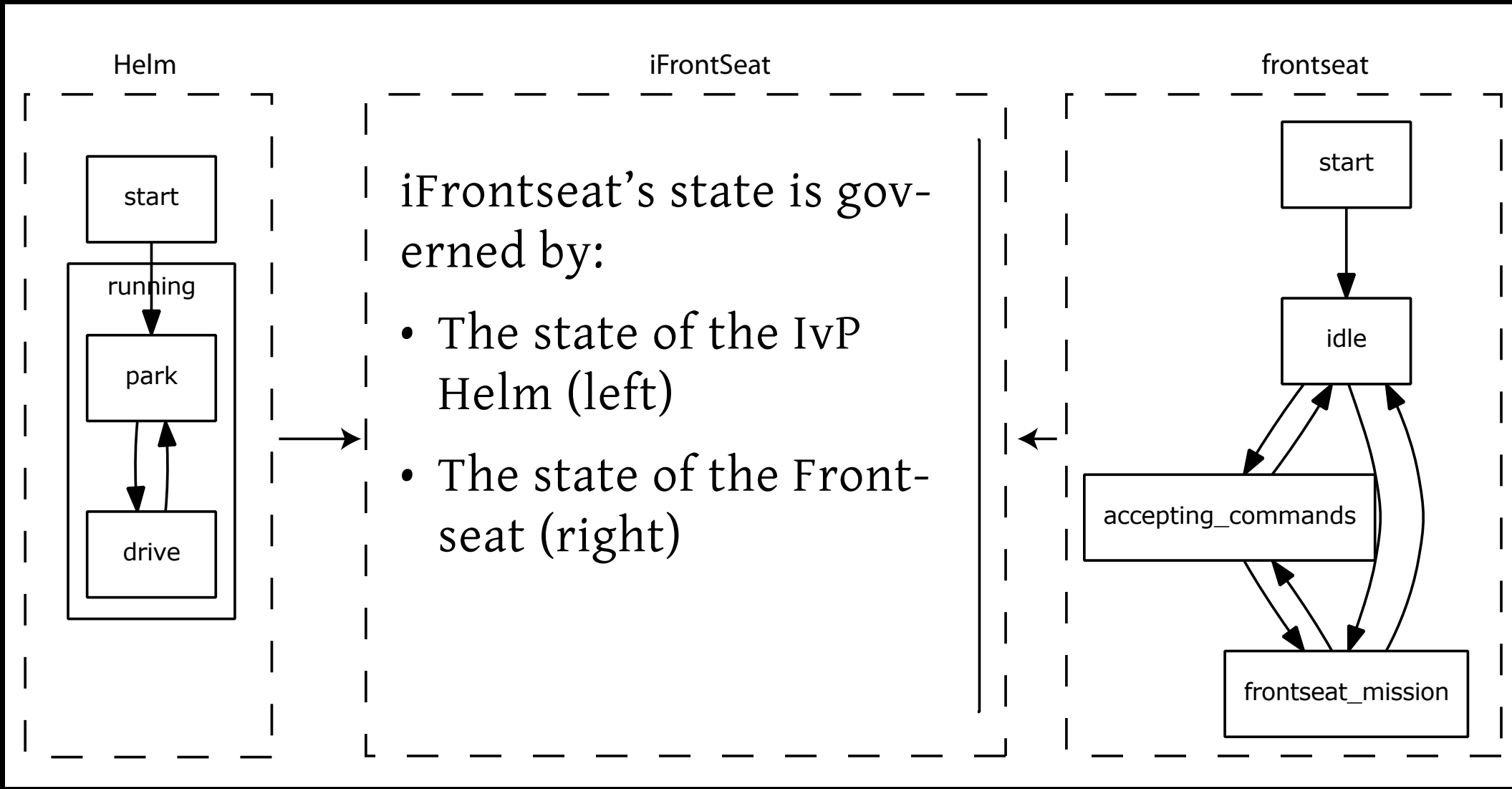
- *Current State*: Maintain knowledge of operational state of both frontseat and backseat.
- *Command*: Interface and translate IvP Helm commands to frontseat
- *Receive data*: from sensors residing on frontseat.
- *Send data*: from sensors residing on backseat to frontseat.



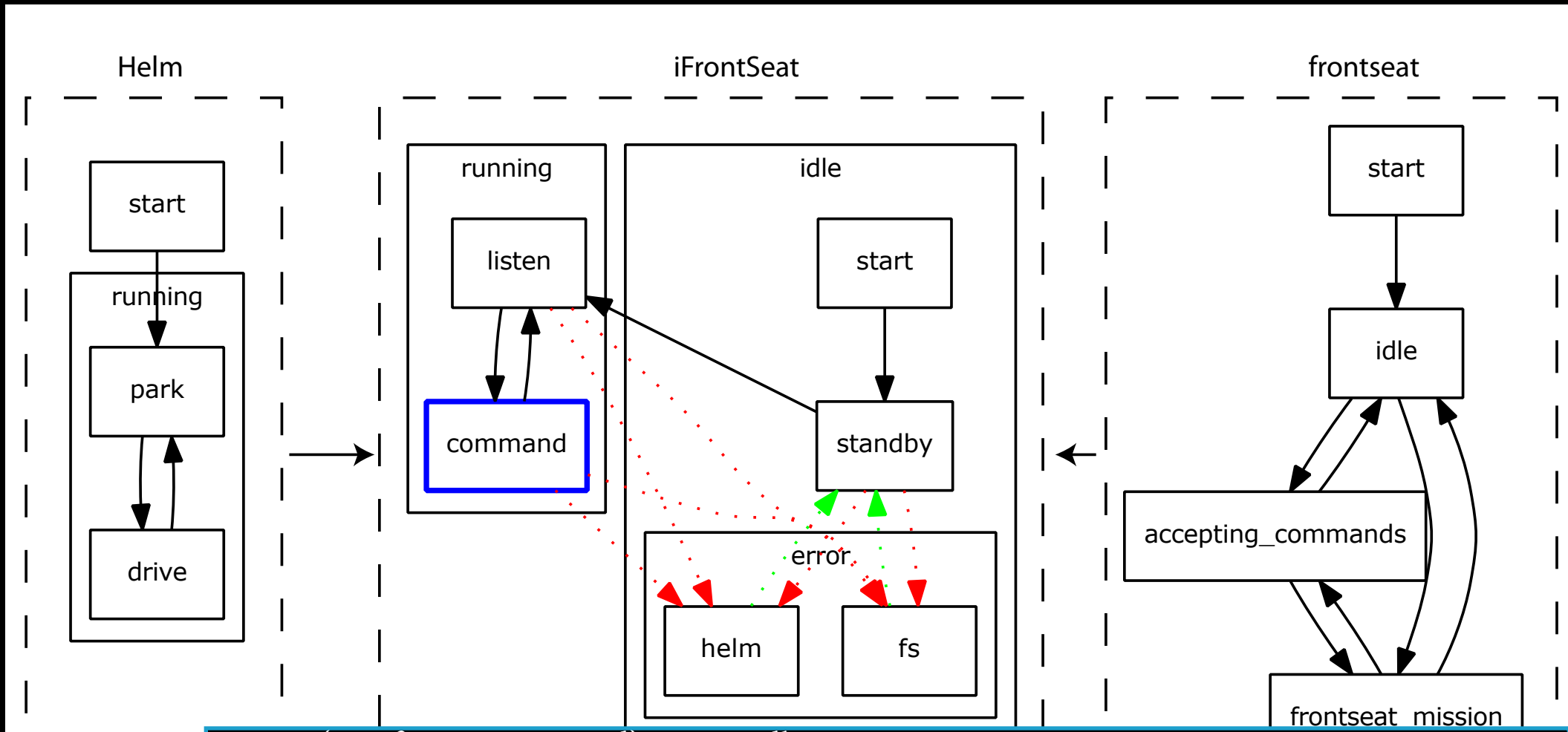
Motivations

- Allow for future expansion for other AUV manufacturers *without rewriting common code.*
  - Support common subset of features
  - Support special vehicle/manufacturer specific features
- C++
  - performance on embedded systems (e.g. ARM)
  - high level abstractions

Backseat (Helm) / Frontseat State



iFrontSeat state



Start --(Configuration Read)--> Standby

Standby --(FS: Providing Data)--> Listen

Listen --(FS: Accepting Commands & Helm: Drive)--> Command

Command --((FS: Frontseat Mission || FS: Idle) & Helm: Drive)--> Listen

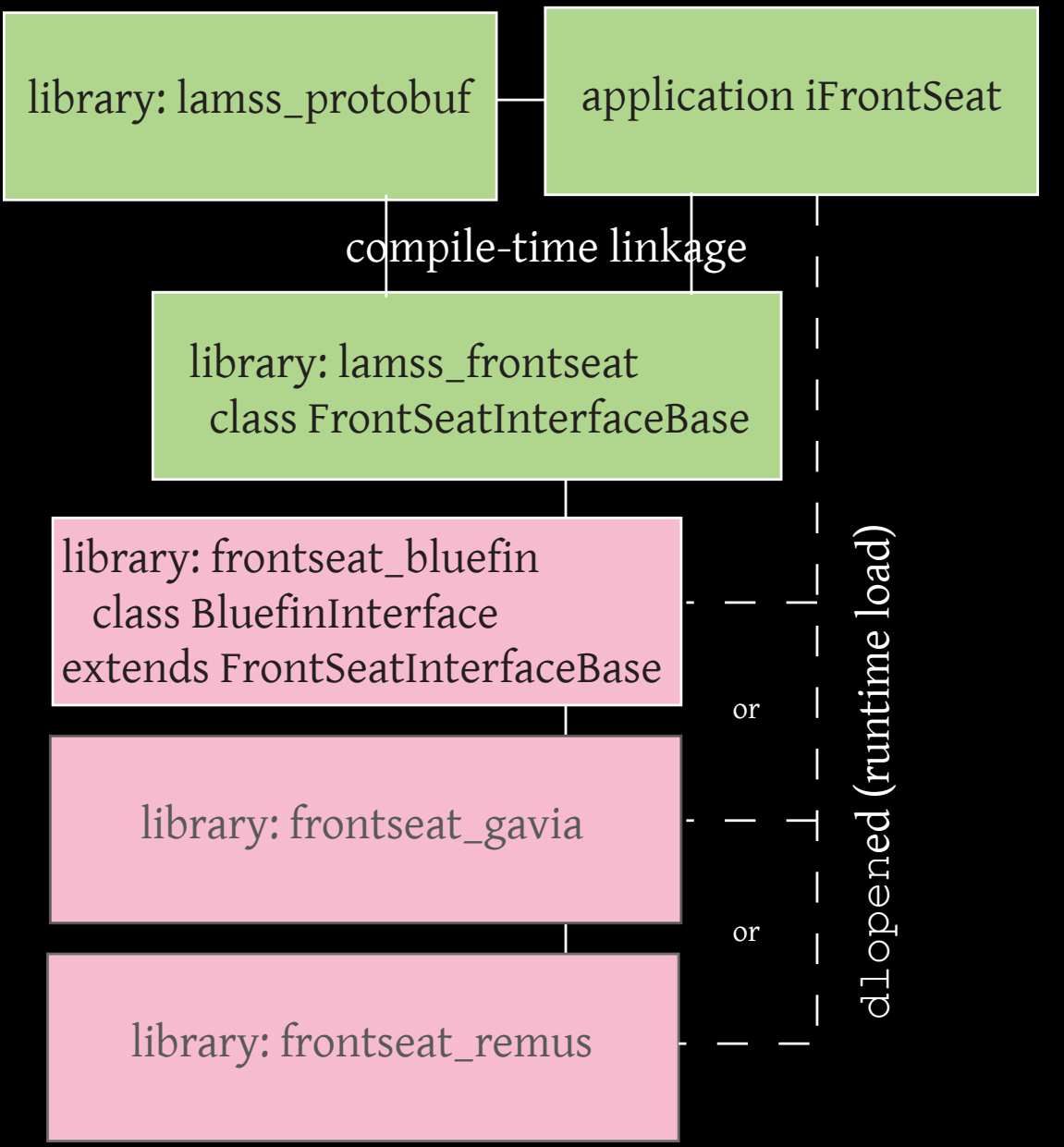
iFrontSeat Static Structure

Some text.

- A bulleted item

Green = Implement Once

Pink = Implement for each vehicle



Main MOOS Interface

Subscriptions (all TextFormat Google Protocol Buffers)

- IFS_COMMAND_REQUEST: Command (such as desired course) to frontseat
- IFS_DATA_TO_FRONTSEAT: Data required by the frontseat from sensors on the backseat.

Publications (all TextFormat Google Protocol Buffers)

- IFS_COMMAND_RESPONSE: Response for a given command
- IFS_STATUS: State of frontseat, Helm, iFrontSeat
- IFS_DATA_FROM_FRONTSEAT: Data from the frontseat (such as navigation).

Transitional MOOS Interface

Translations (type - double or for IFS*: TextFormat Google Protocol Buffers)

- DESIRED_HEADING (degrees), DESIRED_SPEED (m/s), DESIRED_DEPTH (m) parsed and published to IFS_COMMAND_REQUEST.
- CTD_CONDUCTIVITY (S/m), CTD_TEMPERATURE (°C), CTD_PRESSURE (dbar), CTD_SALINITY parsed and published to IFS_DATA_TO_FRONTSEAT
- NAV_* published from IFS_DATA_FROM_FRONTSEAT (if contains navigation object).

Vehicle drivers

Vehicle drivers (e.g. *frontseat::Bluefin, frontseat::Iver*):

- Are a subclass of *frontseat::InterfaceBase* (in liblamss_ frontseat).
- Are compiled into a shared library loaded at runtime by iFrontSeat using the environmental var: IFRONTSEAT_ DRIVER_LIBRARY
- Do not directly deal with MOOS (thus can be developed by non-MOOS C++ programmer)
- Do not have to worry about IvP Helm state
- Do not have read and validate configuration.
- No contraints are placed on the frontseat transport (TCP, serial, UDP, etc.) or data serialization (NMEA-0183, binary, custom).

frontseat::Bluefin

Implements the *frontseat::InterfaceBase* for the Bluefin Robotics Standard Payload Interface v1.8 (and older?)

- Bluefin uses NMEA-0183 over TCP
- CTD and Micro-Modem may be connected to either frontseat or backseat.
- Handles large number of special features (buoyancy control, trim control, etc.) through extensions of the core command message.

Lessons:

- Well defined backseat interface can lead to better frontseat interface implementation ($BFCTL).
- Don’t assume vehicles from same manufacturer will have same frontseat software.

frontseat::Iver

Implements the *FrontSeatInterfaceBase* for the OceanServer Iver "Remote Helm" over serial.

Uses:

- $OSI for lat, lon, speed, mission state, depth, altitude, heading
- $C for pitch, roll
- $OMS for desired heading, speed, depth

Writing a new driver

Implement virtual methods of FrontSeatInterfaceBase:

- send_command_to_frontseat
- send_data_to_frontseat
- frontseat_state
- frontseat_providing_data
- send_raw_to_frontseat

Process frontseat messages and post signals:

- signal_command_response
- signal_data_from_frontseat
- signal_raw_from_frontseat
- signal_raw_to_frontseat

iFrontSeat configuration

"iFrontSeat -e" will always give all valid configuration

```
ProcessConfig = iFrontSeat
{
  common {
  // ...
  }
  frontseat_cfg {
  // ... configuration for the frontseat driver plugin
  }
  moos_var {
  // ... configuration of the MOOS variable names (usually omitted)
  }
  legacy_cfg {
  // ... configuration for legacy pub/subs
  }
}
```

iFrontSeat configuration

```
ProcessConfig = iFrontSeat_basic_simulator
{
    frontseat_cfg {
        require_helm: true
        helm_running_timeout: 10
        frontseat_connected_timeout: 10
        status_period: 5
        exit_on_error: false
        type: AUV
        [goby.middleware.frontseat.protobuf.basic_simulator_config] {
            tcp_address: "127.0.0.1"
            tcp_port: 54321
            start {
                lat: 41.59 # deg
                lon: -70.71 # deg
                duration: 0 # 0 == no timeout, otherwise timeout in seconds
                control_freq: 10 # Hz
                vehicle {
                    accel: 1.0 # m/s^2
                    hdg_rate:  80 # deg/s
                    z_rate: 2 # m/s
                }
            }
        }
    }

}
```