Update on the adoption of synchronous languages at gh.st

Kai Engelhardt

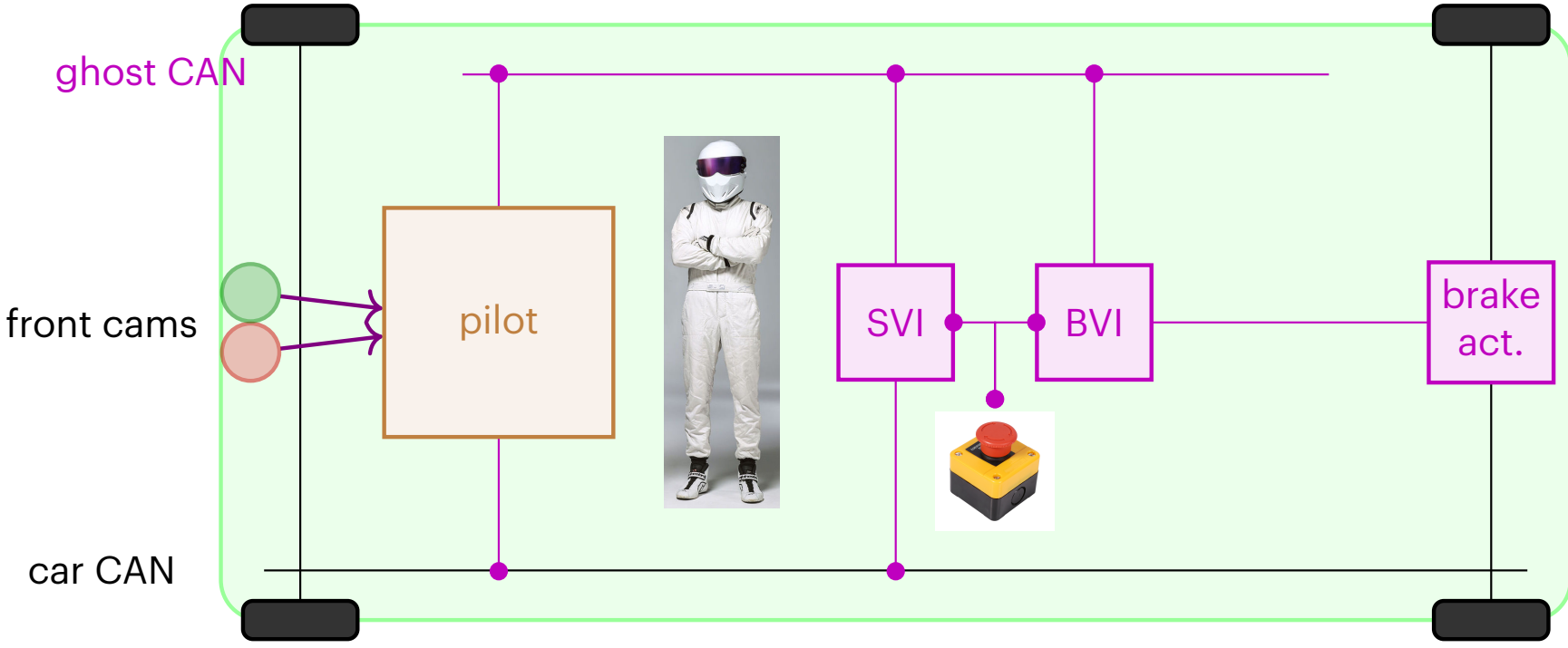
views expressed, if any, are mine, not Ghost's

What is gh.st?

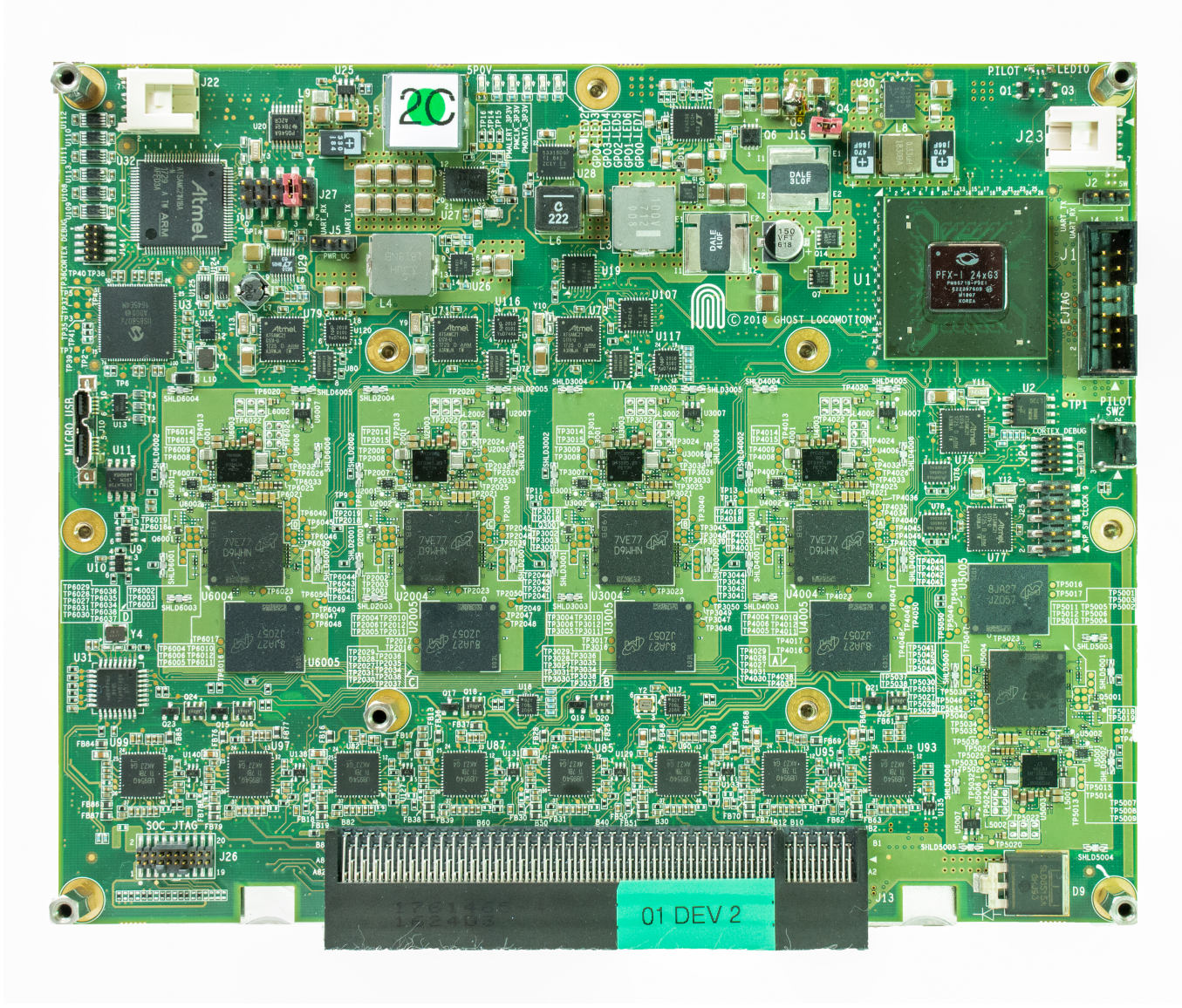
Now: Build hardware and train AI to drive a car in good conditions on a Californian highway.

Then: Bootstrap by recording more driving so the AI can learn to drive in more places and conditions.

Prototype



Pilot Board



SW stack overview

Driving decision computation is described in our own language, GFL.
GFL has two sets of operators

- streaming operators (functional data flow with bounded history)
- neural network operators (convolution, ReLU, ...)

Now: GFL is realised as an interactive system; it's a Scala EDSL, translates to byte code that runs in a JVM on Android/linux; NN ops are translated to OpenCL so they can run on GPUs

Then: re-interpret GFL as reactive system; translate to Lustre (for streaming ops) + FFI (for NN ops); run on (almost) bare metal; non-critical bits remain in Scala-land

Main Goal of Demo Validation

Have **SVI**, **BVI**, and **brake actuator** as a simple and reliable **insulation layer** between

untrusted NNs, GFL, Android, linux, 845s

---

trusted car, safety driver

Demo Safety

Safe disengagement: once disengaged, the Ghost Camry behaves like a Camry

Instantaneous disengagement: any trouble leads to immediate disengagement (where "trouble" is detectable for the SVI)

No spontaneous engagement: only when all planets are aligned…

Don't write car CAN: at most read the car CAN

Pilot doesn't talk to BVI: only SVI instructs BVI

SVI vets driving decisions: carry out only timely driving decisions that fit the current envelope

Altogether: **about as safe as without ghost**

Techniques for Demo Validation

E-Stop & brake act. & SVI car CAN μC: EE (and ME) so ghost surgery doesn't corrupt the car

SVI & BVI & pilot CAN μCs: Testing, PBT, model checking for Lustre and C

deployment: pilot's check list incl. manual validation of firmware versions on demo car

What about the NNs?

The NNs can be `rand()` without jeopardising demo safety.

A **successful** demo needs ghost to drive for a while.
And that's what we did on the 280!

Experiment: Lustre for μC

We rewrote core vehicle interface functionality in Lustre (brake, steering, accelerator)

Now: μC code for FSMs, calls C to drive μC internals
30 `.lus` files, 2.8k loc, 94 properties
26k loc `.[ch]` (incl. μC boilerplate)

- language dialect differences bridged with in-house Lustre babelfish (`tril`)
- compiled with heptagon (after `.lus` $\xrightarrow{\texttt{tril}}$ `.ept`)
- 80 properties verified with `kind2`

Then:

- move to VÉLUS
- add theorem-prover escape hatch to state and prove properties beyond `kind2`
- ASIL-D-certified μC where redundancy is inopportune

Example

```
-- calculate delta since last clock
t_last = 0 -> pre t_now;
t_delta = if t_now > t_last then t_now - t_last
  else t_now + (INT_MAX - t_last);
t_good = t_delta >= T_MIN_US and t_delta <= T_MAX_US;

-- input voltage is good if its DC value is in range
v_clamp = clamp(0, EMA_RANGE, v_input);
v_filt = ema50(v_clamp);
v_good = v_filt >= V_INPUT_MIN_DC and v_filt <= V_INPUT_MAX_DC
    and v_input >= V_INPUT_MIN;

fault = if not lastn(adc_clk, GOOD_CLOCKS_MIN) then DISENG_SYS_ADC
  else if not lastn(v_good, GOOD_CLOCKS_MIN) then DISENG_SYS_VIN
  else if not lastn(t_good, GOOD_CLOCKS_MIN) then DISENG_SYS_OVERRUN
  else DISENG_NONE;

--%PROPERTY (t_now - t_last) >= 9000 and (t_now - t_last) <= 11000 => t
--%PROPERTY t_now > t_last and t_now - t_last < 9000 => not t_good;
--%PROPERTY not (adc_clk or v_good or t_good) => faulty(fault);
```

Results

- Steering interface (`.lus`) featured in a driving demo on highway 280 in CA. Braking and acceleration were not demoed on public roads yet.
- Engineers working on the μCs, even those with zero previous exposure to FM, took to writing `.lus` and simple properties.
- The μCs work as expected!
  We found **one** bug survived in our Lustre code:

  > **Bug:** A race between sampling different analog pins and executing the lustre `tick` function; analog pins containing redundant information were sampled at different times, leading to spurious faults due to the redundant pins not agreeing

- `kind2` is somewhat weak when it comes to numerics and automata; working around that, we used it quite successfully

Experiment: SCCharts for FSMs

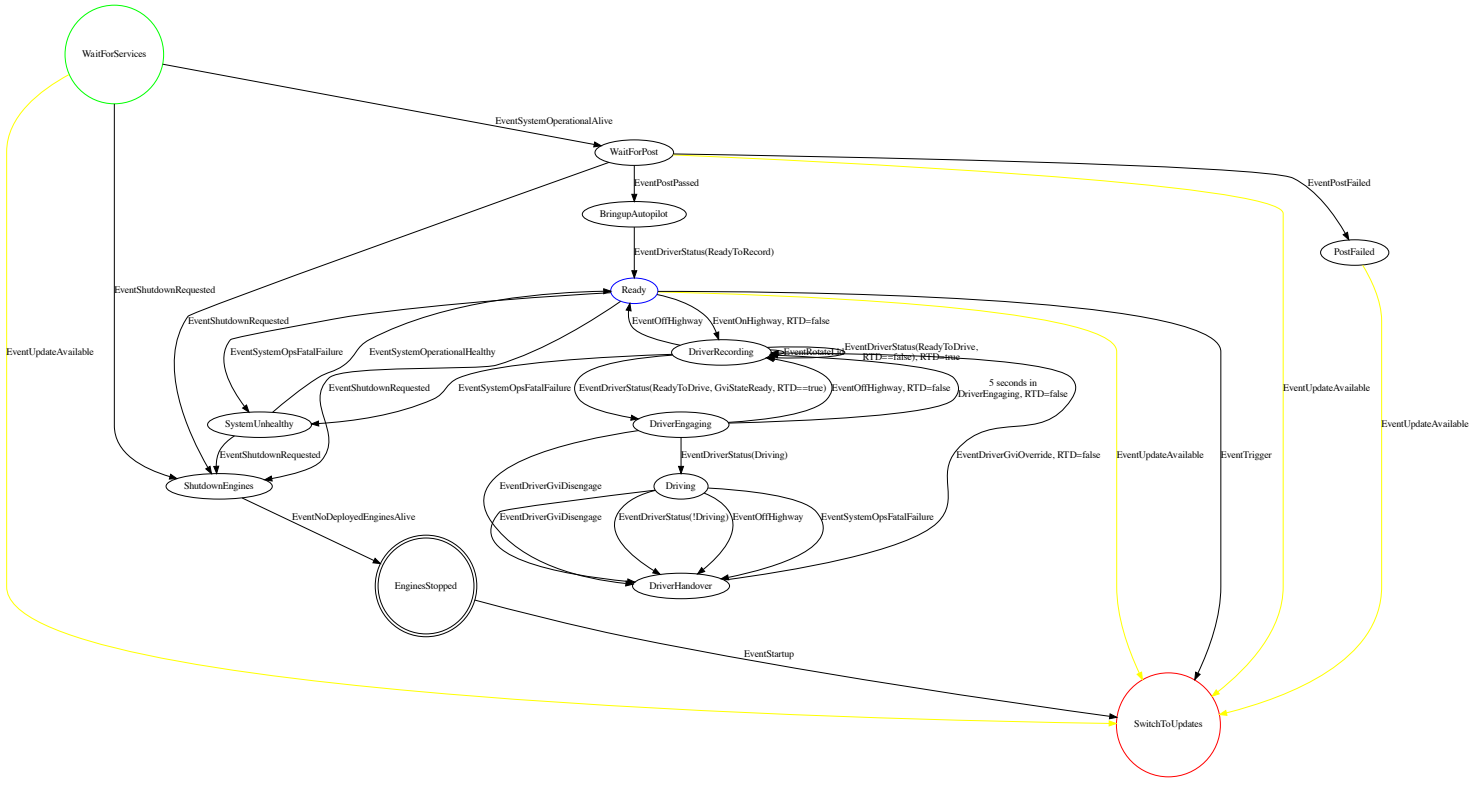
Now: We rewrote pilot FSMs in SCCharts
(HMI, coordinator, and driver FSMs which run on all SoCs)

- exported to Java
- linked to Scala via JNI
- runs on a JVM/Android/linux

Then:

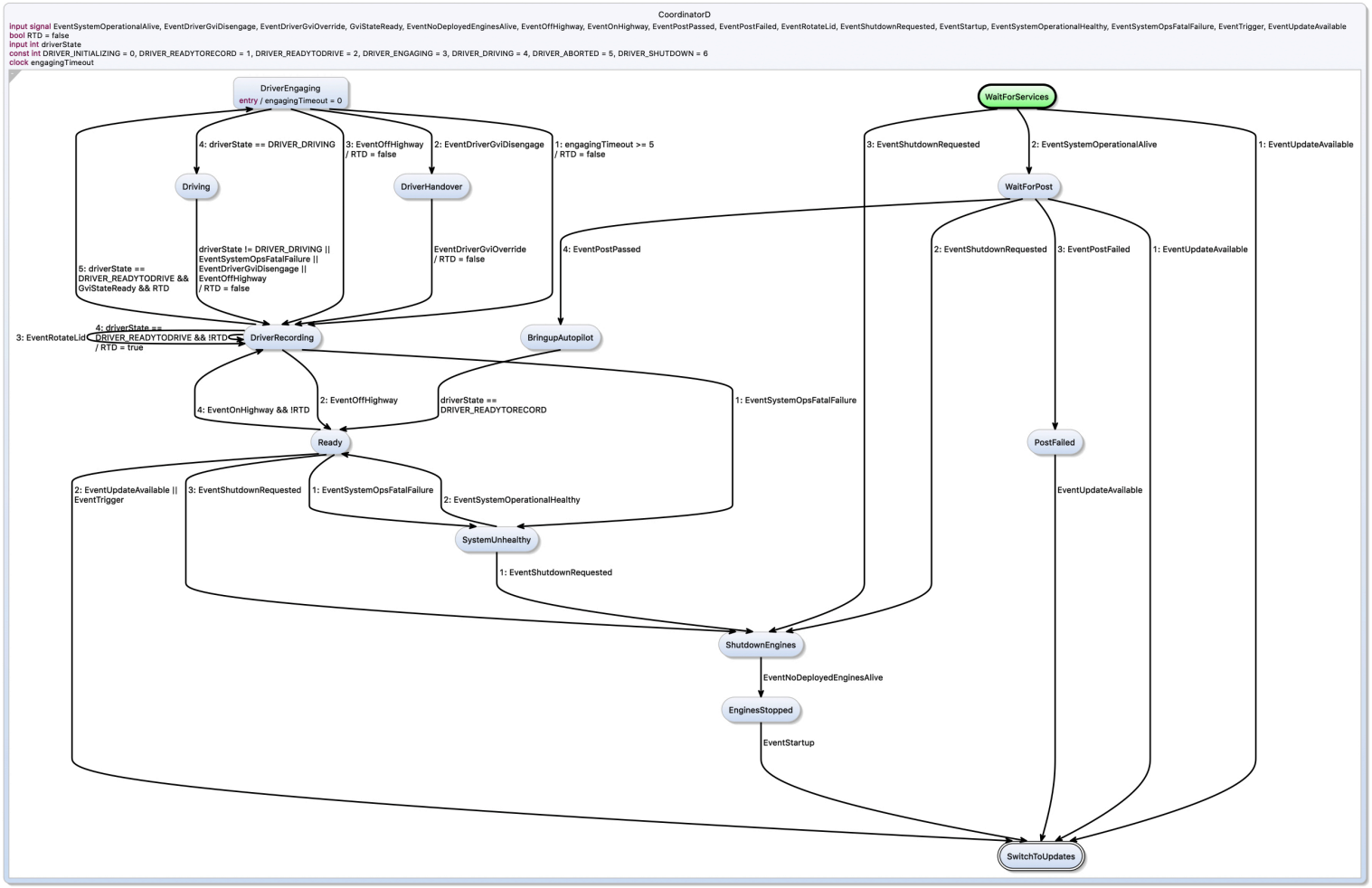
- run on hypervisor/HAL instead of JVM
- rewrite or export to Lustre to use Vélus

Example: Coordinator FSM



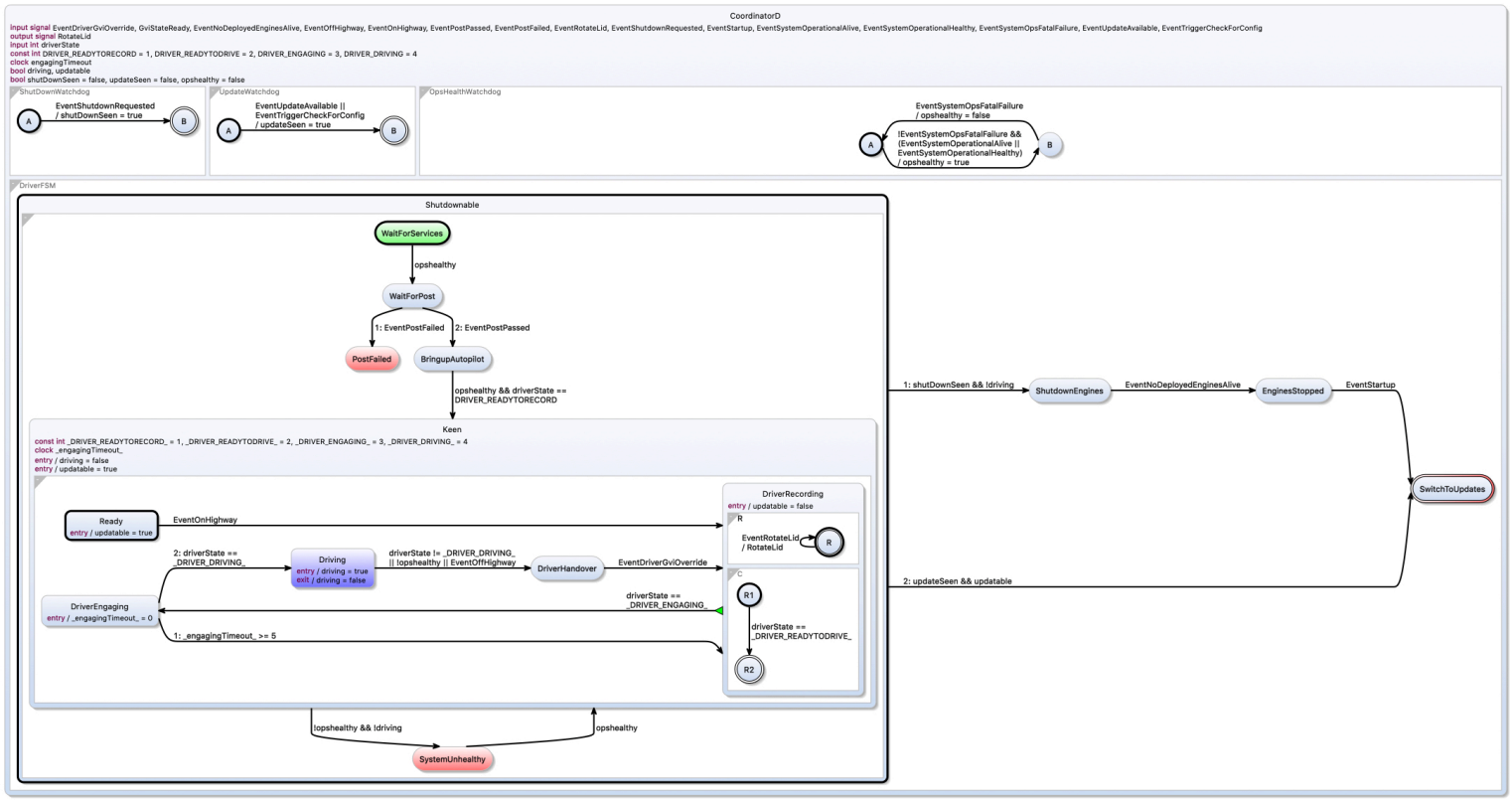
part of the coordinator, in `.dot` derived from Akka/Scala

Example: Coordinator FSM



same part, redone as SCChart

Example: Coordinator FSM



same, cleaned up and repaired

Results

- The driver FSM (`.sctx`) featured in the same driving demo on highway 280 in CA.
- Engineer working on the driver FSMs took to writing `.sctx`.
- Simulating individual FSMs in Kieler helped communicating, e.g. concerns about corner cases.
- Eliminating scheduling conflicts can be hard.
- We found logic bugs, missing transitions, and a deadlock.

Misc. FM

- `Why3` to model some of the OpenCL tensor operators used by GFL; WhyML $\rightarrow$ OpenCL extractor
- `cbmc` to validate power-μC functionality
- `ghedrc` (in-house dev.) reads a schematic in Allegro netlist format and performs structural and semantic electrical checks