Unified Graphical Co-modelling, Analysis and Verification of CPSs by combining AADL and Simulink/Stateflow

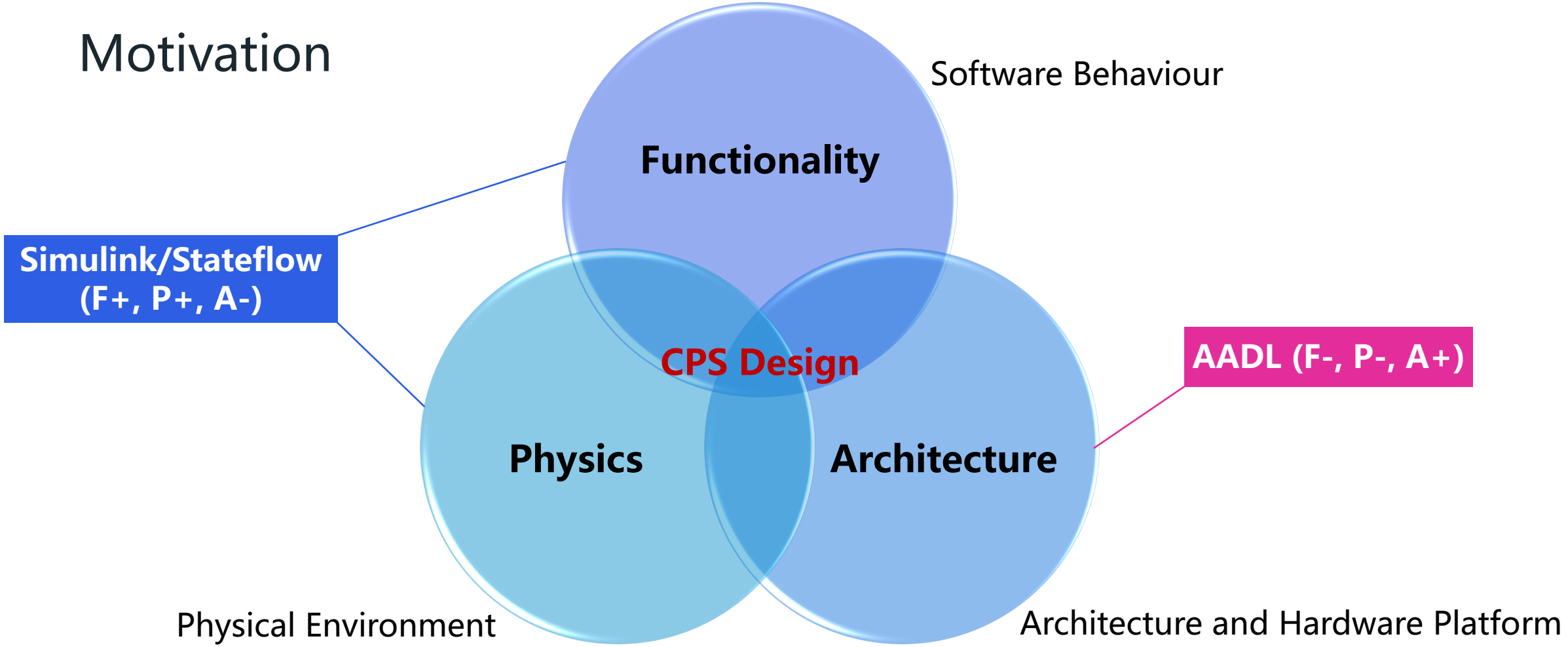
Xiong Xu[1,3], Shuling Wang[1], Bohua Zhan[1,2], Xiangyu Jin[1,2]

Jean-Pierre Talpin[3], Naijun Zhan[1,2]

[1]*Institute of Software, CAS* [2]*University of CAS* [3]*Inria*

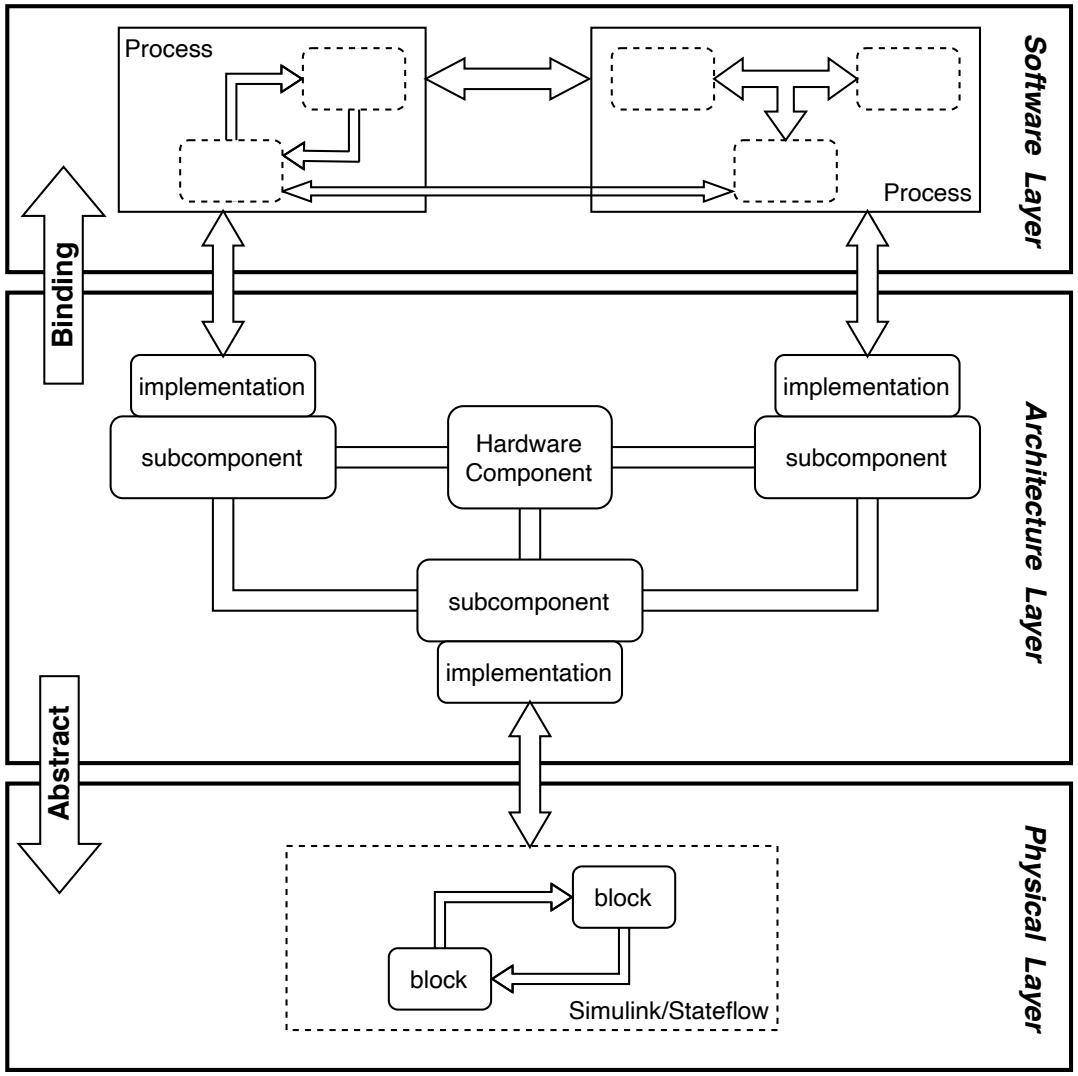
The ADEPT workshop, June 17, 2022

Motivation



AADL⊕Simulink/Stateflow	A unified graphical co-modelling formalism supporting the design of CPSs from all these three dimensions uniformly	**F+P+A**

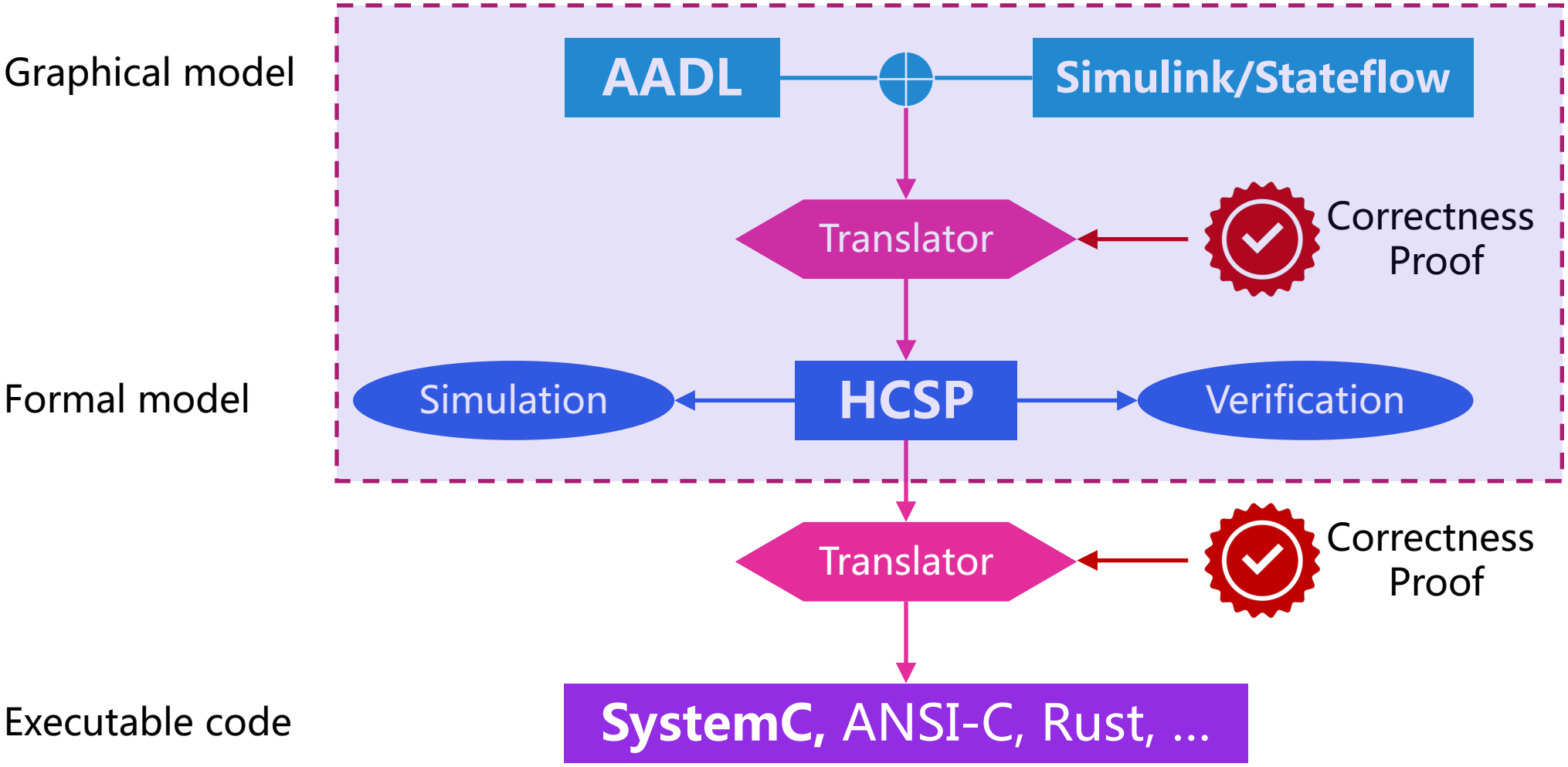
Overview of AADL⊕S/S




```
thread PI_ctr
    features
        veh_v: in data port;
        des_v: in data port;
        des_a: out data port;
    properties
        dispatch_protocol => periodic;
        period => 7ms;
        compute_deadline => 7ms;
        compute_execution_time => 1ms;
        priority => 1;
end PI_ctr

thread implementation PI_ctr.imp
    annex Simulink{** ./PI_ctr_imp.xml **};
end PI_ctr.imp
```

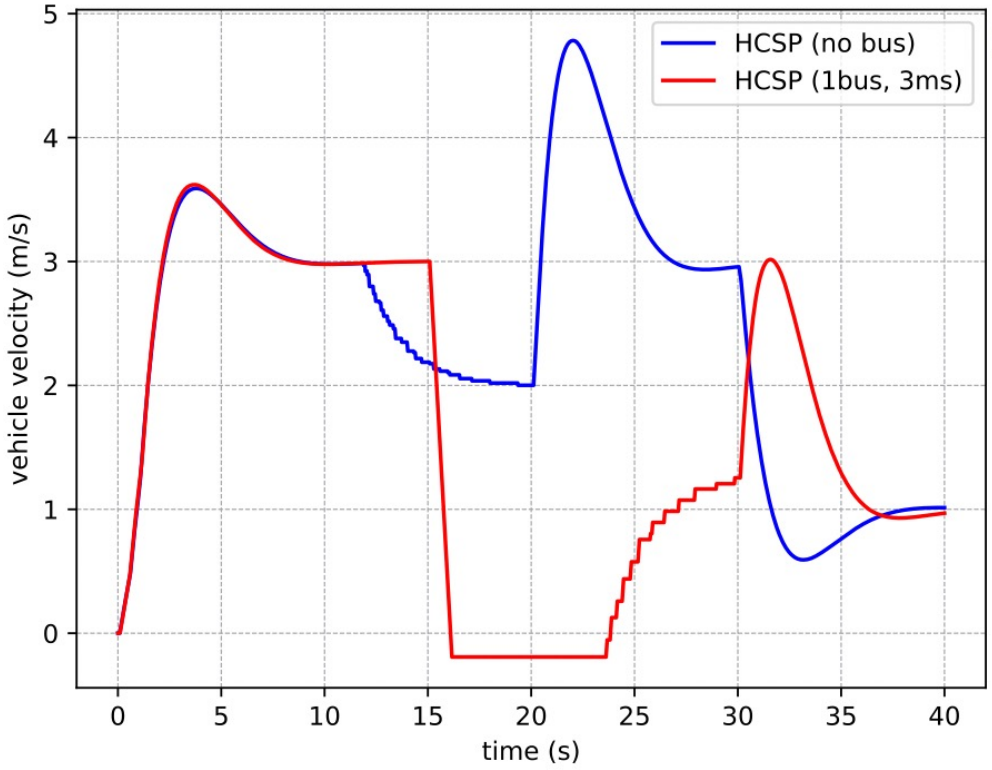
```
abstract vehicle
    features
        veh_a: in data port;
        veh_s: out data port;
        veh_l_v: out data port;
        veh_w_v: out data port;
end vehicle;

abstract implementation vehicle.imp
    annex Simulink{** ./vehicle_imp.xml **};
end vehicle.imp;
```

MARS: a toolkit for **M**odelling, **A**nalysis, and v**R**ification of hybrid **S**ystems



HCSP: Hybrid Communicating Sequential Processes

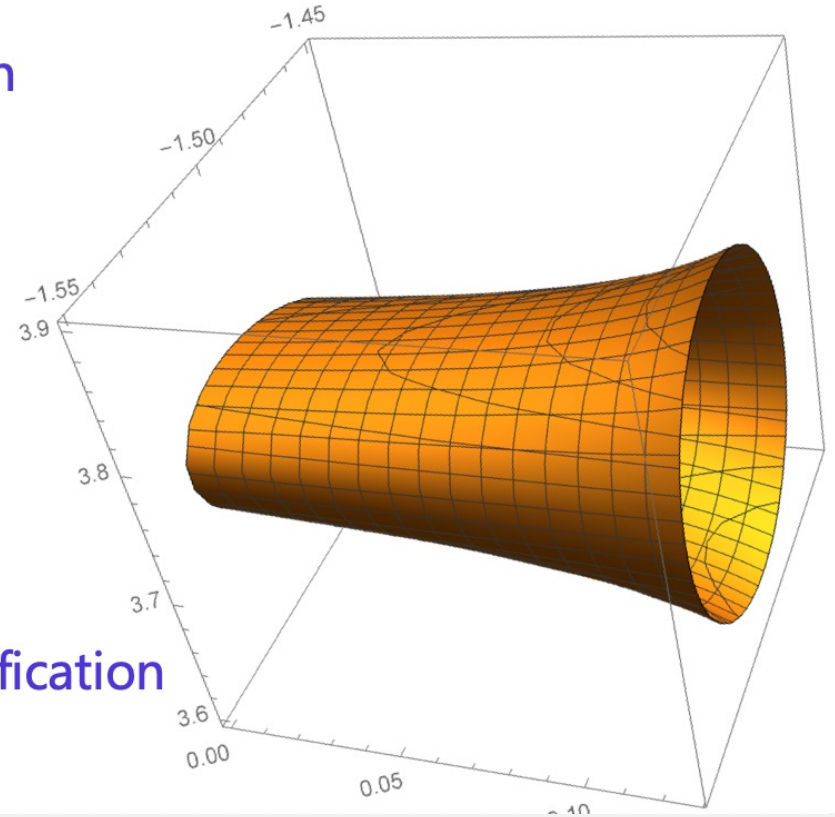
$$
\begin{aligned}
P &::= \text{skip} \mid x := e \mid ch?x \mid ch!e \mid P; Q \mid B \to P \mid P \sqcap Q \mid P^* \mid [\!]_{i \in I}(io_i \longrightarrow Q_i) \mid \\
&\quad \langle F(\dot{\mathbf{s}}, \mathbf{s}) = 0 \& B \rangle \mid \langle F(\dot{\mathbf{s}}, \mathbf{s}) = 0 \& B \rangle \trianglerighteq [\!]_{i \in I}(io_i \longrightarrow Q_i) \\
S &::= P_1 \| P_2 \| \ldots \| P_n \text{ for some } n \geq 1
\end{aligned}
$$



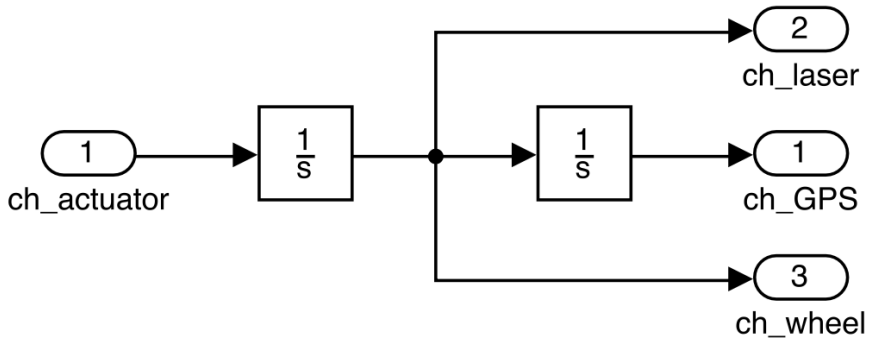
Simulation

&

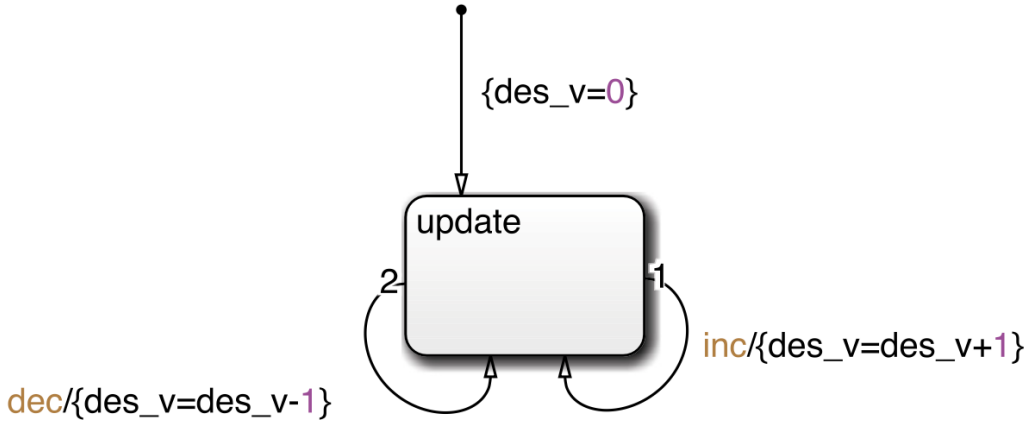
Verification



From Simulink/Stateflow to HCSP



(a) `vehicle.imp`



(b) `pan_ctr_th.imp`

$$
\begin{array}{l}
v := 0; s := 0; \\
sent_{laser} := \textbf{false}; sent_{wheel} := \textbf{false}; sent_{GPS} := \textbf{false}; \\
\textbf{while } \neg(sent_{laser} \wedge sent_{wheel} \wedge sent_{GPS}) \\
\qquad [] \begin{pmatrix} ch_{laser}!v & \longrightarrow & sent_{laser} := \textbf{true} \\ ch_{wheel}!v & \longrightarrow & sent_{wheel} := \textbf{true} \\ ch_{GPS}!s & \longrightarrow & sent_{GPS} := \textbf{true} \end{pmatrix} \\
\textbf{endwhile}; ch_{acturator}?a; \\
\left(\langle \dot{s} = v, \dot{v} = a \& \textbf{true} \rangle \trianglerighteq [] \begin{pmatrix} ch_{laser}!v & \longrightarrow & \textbf{skip} \\ ch_{wheel}!v & \longrightarrow & \textbf{skip} \\ ch_{GPS}!s & \longrightarrow & \textbf{skip} \\ ch_{acturator}?a & \longrightarrow & \textbf{skip} \end{pmatrix} \right)^*
\end{array}
$$

$$
\begin{array}{l}
des\_v := 0; \\
\left(\begin{array}{l} inputs?event; \\ event == \text{"inc"} \rightarrow des\_v := des\_v + 1; \\ event == \text{"dec"} \rightarrow des\_v := des\_v - 1; \\ outputs!des\_v \end{array} \right)^*
\end{array}
$$

L. Zou, et al. **Verifying Simulink diagrams via a hybrid Hoare logic prover**. *EMSOFT* 2013, pp. 1-9.

N. Zhan, et al. **Formal Verification of Simulink/Stateflow Diagrams: A Deductive Approach**. *Springer*, 2017.

From AADL to HCSP (Scheduler and Dispatcher)

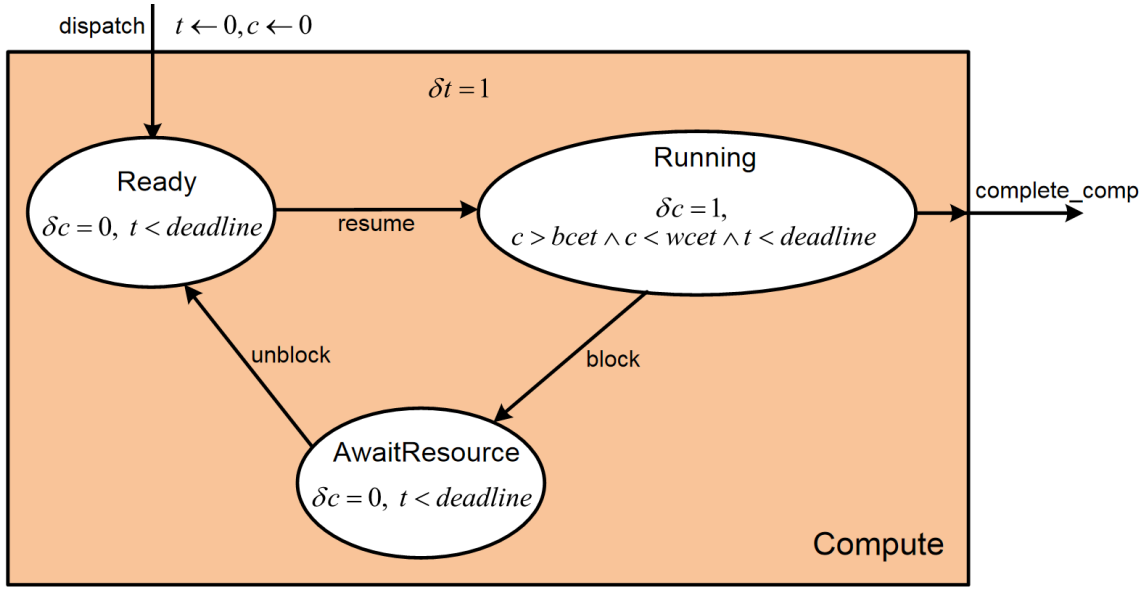
```
module SCHEDULE_HPF(sid) ::=
  Pool := [];
  run_now := -1; run_prior := -1
  (
  reqProcessor[sid][_tid]?prior ⟶
      if run_prior > prior then
          Pool := put(Pool, [prior, _tid])
      else
          run_now != -1 → preempt[sid][run_now]!;
          run_now := _tid;
          run_prior := prior;
          run[sid][run_now]!
      endif
  □ free[sid][_tid]? ⟶
      assert(_tid == run_now);
      if len(Pool) > 0 then
          (run_prior, run_now) := get_highest(Pool)
          Pool := delete(Pool, run_now);
          run[sid][run_now]!
      else
          run_prior := -1; run_now := -1
      endif
  □ exit[sid][_tid]? ⟶ Pool := delete(Pool, _tid)
  )*
```

```
module DIS_aperiodic(send, out_port, recv, in_port) ::=
  queue := [];
  (
  if len(queue) == 0 then
      outputs[send][out_port]?event;
      queue := push(queue, event)
  else
      outputs[send][out_port]?event ⟶
          queue := push(queue, event)
      □ dis[recs][in_port]!head(queue) ⟶
          queue := tail(queue)
  endif
  )*
```

```
module DIS_periodic(tid, period) ::=
  (
  wait(period);
  dis[tid]!
  )*
```

- Processor ⇒ Scheduler (FIFO, RMS, DMS, HPF)
- Dispatchers are for threads

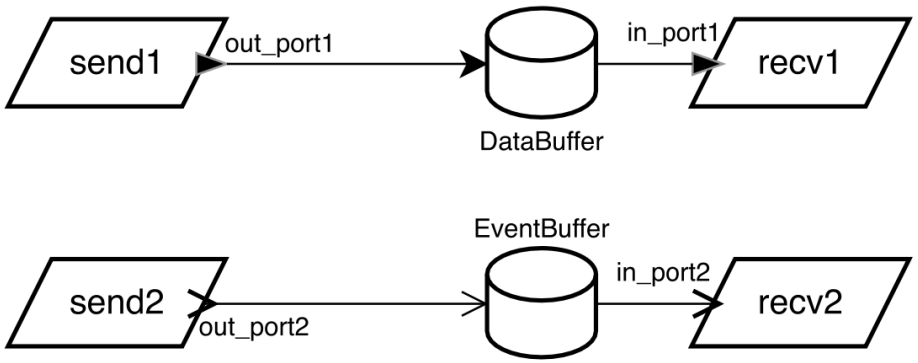
From AADL to HCSP (Thread)



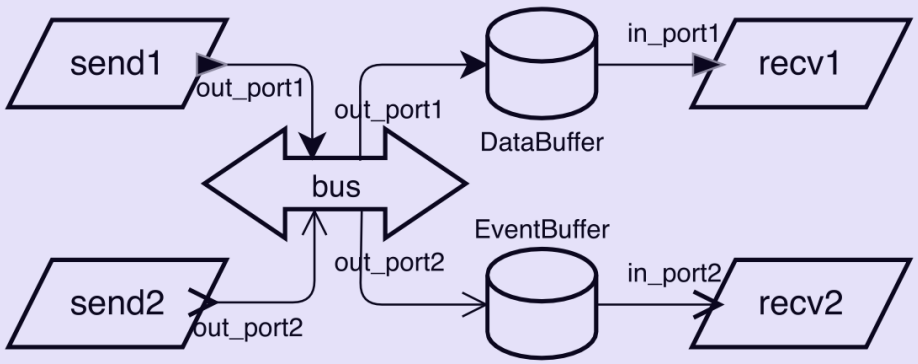

```
module EXE(tid, prior, WCET, deadline) ::=
  @Initialisation;
  state := “wait”
  (
  if state == “wait” then
      dis[tid]?; @Input;
      t := 0; comp_complete := 0; state := “ready”
  elif state == “ready” then
      reqProcessor[tid]!prior;
      <t_dot = 1 & t < deadline>⊵
          □(run[tid]? ⟶ state := “running”);
      state == “ready” →
          exit[tid]! ⟶ state := “wait”
          □run[tid]? ⟶ state := “running”
  elif state == “running” then
      @RUNNING
  else  # state == “await”
      <t_dot = 1 & t < deadline>⊵
          □(unblock[tid]? ⟶ state := “ready”);
      state == “await” → state := “wait”
  endif
```

```
procedure RUNNING ::=
  if comp_complete == 0 then
      c := 0; @DiscreteCompute; comp_complete := 1
  else
      <t_dot = 1, c_dot = 1 & c < WCET ∧ t < deadline> ⊵
          □preempt[tid]? ⟶ state := “ready”
      state == “running” →
          if c == WCET then
              @Output
          else
              preempt[tid]? ⟶ state := “wait”
              □free[tid]! ⟶ state := “wait”
          endif
  endif
```

From AADL to HCSP (Connection)



(a) Buffers between ports



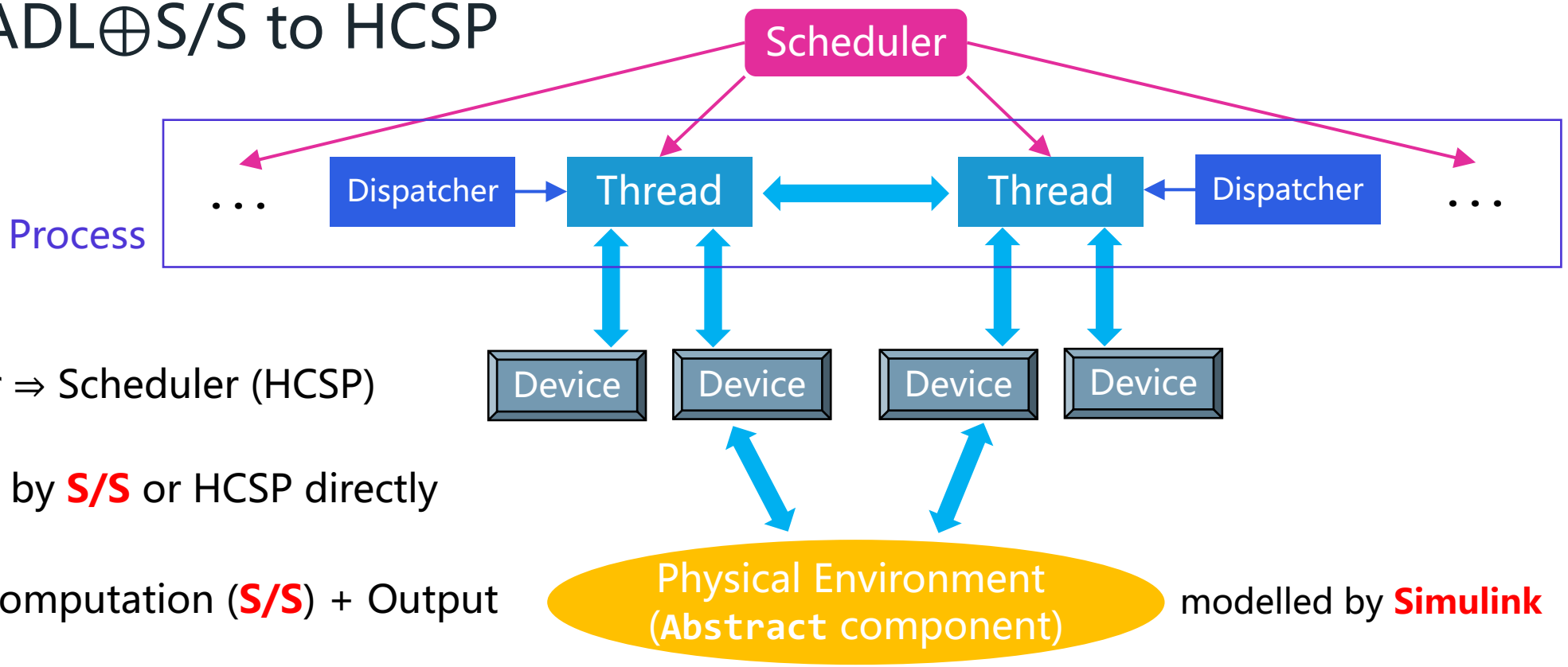
(b) Two sender threads share one bus

```
module BUS(bus_id, send1, out_port1, send2, out_port2) ::=
  (
  reqBus[send1]? ⟶
      outputs[send1][out_port1]?data;
      @BLOCK2;
      outputs[bus_id][out_port1]!data
   ▯ unblock[send1]! ⟶ skip
   ▯ reqBus[send2]? ⟶
      outputs[send2][out_port2]?event;
      @BLOCK1;
      outputs[bus_id][out_port2]!event
   ▯ unblock[send2]! ⟶ skip
  )*
```

```
procedure BLOCK1 ::=
  t := 0;
  while t < latency do
      <t_dot = 1 & t < latency> ⊵
          ▯ block[send1]! ⟶ skip
  endwhile
```

```
procedure BLOCK2 ::=
  t := 0;
  while t < latency do
      <t_dot = 1 & t < latency> ⊵
          ▯ block[send2]! ⟶ skip
  endwhile
```

From AADL⊕S/S to HCSP



Scheduler — Processor ⇒ Scheduler (HCSP)

Device — Modelled by **S/S** or HCSP directly

Thread — Input + Computation (**S/S**) + Output

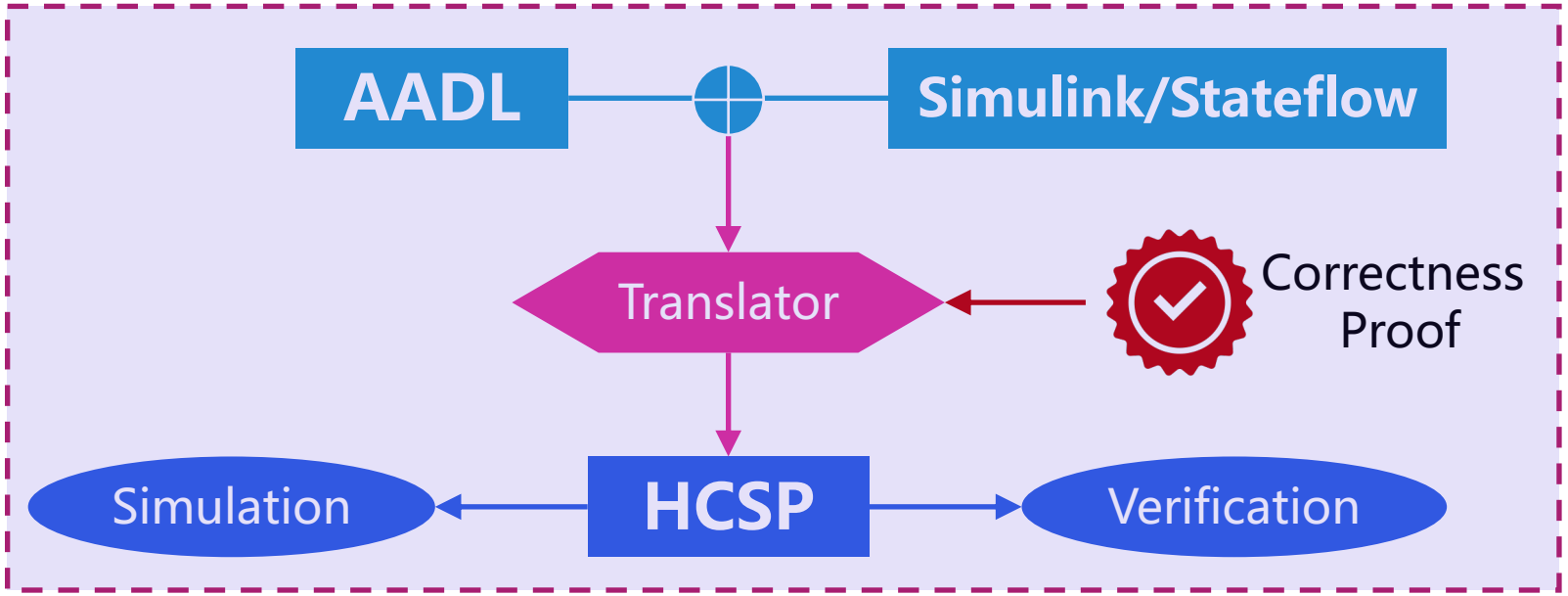
Connections (with or without bus) ⇒ (Data or Event) Buffer (HCSP)

System ::= Scheduler || Dispatchers || Threads || Devices || Connections || Physical

A system is composed by the communication of its components

MARS: From AADL⊕S/S to HCSP

- ✓ Co-modelling
  - AADL
  - Simulink/Stateflow
- ✓ Hybrid annex
  - Simulink/Stateflow
  - HCSP
- ✓ Scheduling analysis
  - HPF scheduler
- ✓ Latency analysis
  - Bus latency
- ✓ Safety analysis
  - Simulation
  - Verification
- ✓ Verification
  - Hybrid Hoare Logic



- ✗ Other AADL components and features, such as Memory
- ✗ Other scheduling policies, such as FIFO, RMS, DMS
- ✗ Verification for complex models
- ✗ …

Correctness of translation

Definition of weak bisimulation

Let $T_i = \langle S_i, A_i, \rightarrow_i \rangle$ be two transition systems for $i = 1, 2$. A relation $R \subseteq S_1 \times S_2$ is a weak bisimulation, iff

- R is symmetric
- $\forall (s, t) \in R$, s and t are equivalent under a variable mapping from T_1 to T_2
- if $s \Rightarrow_1^a s'$ holds, then there exists a path $t \Rightarrow_2^a t'$ such that $(s', t') \in R$

Correctness of translation

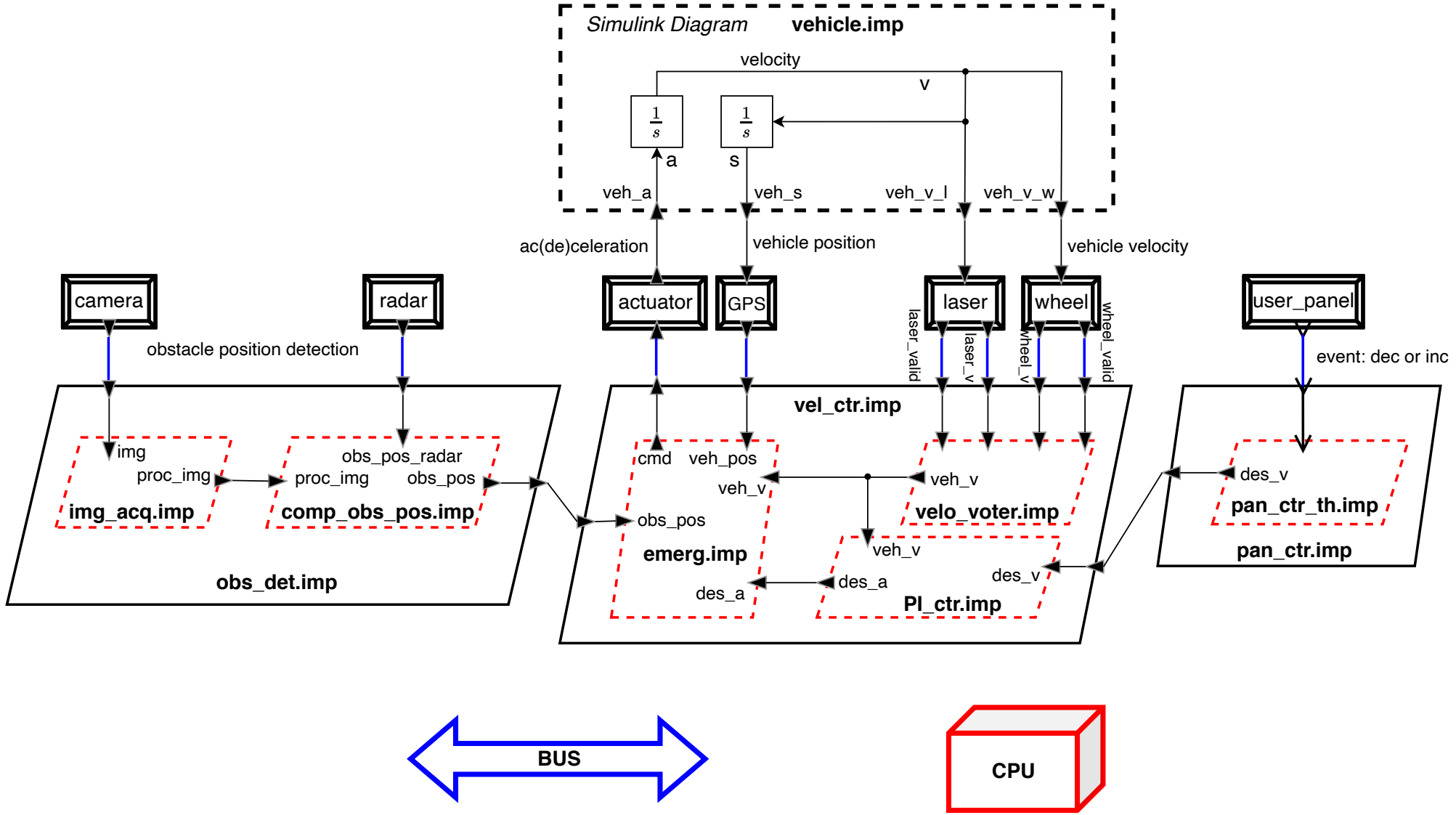
The translation from AADL⊕S/S to HCSP is correct, i.e., there exists a weak bisimulation relation between AADL⊕S/S and its translated HCSP.

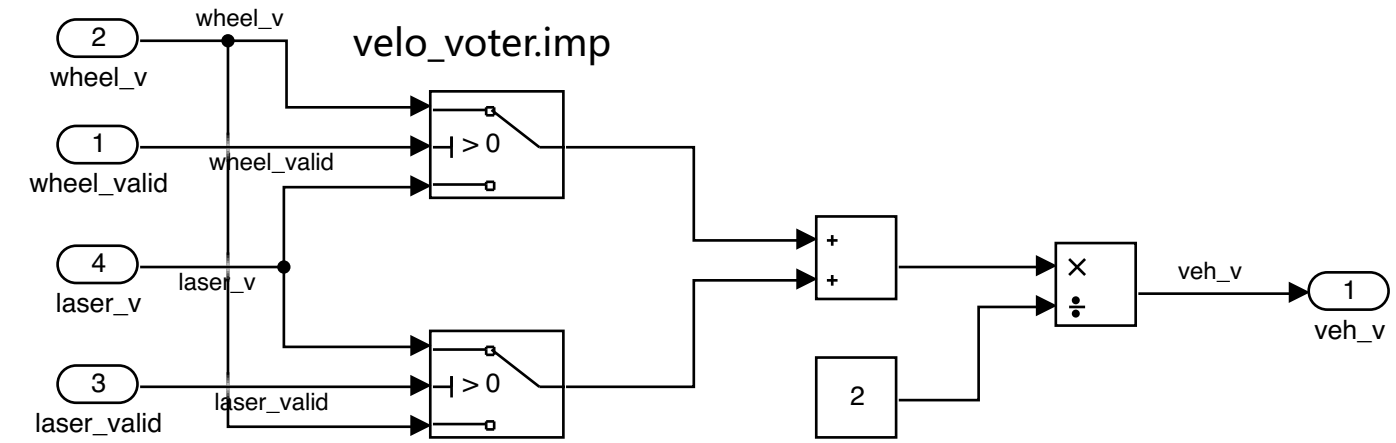
Case study: a cruise control system

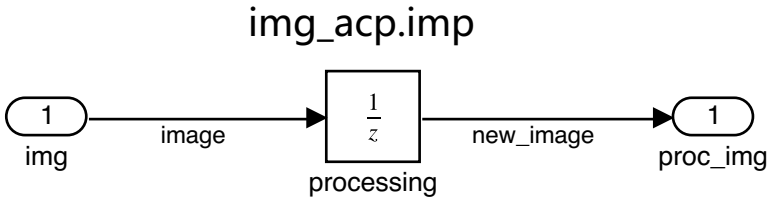
Physical

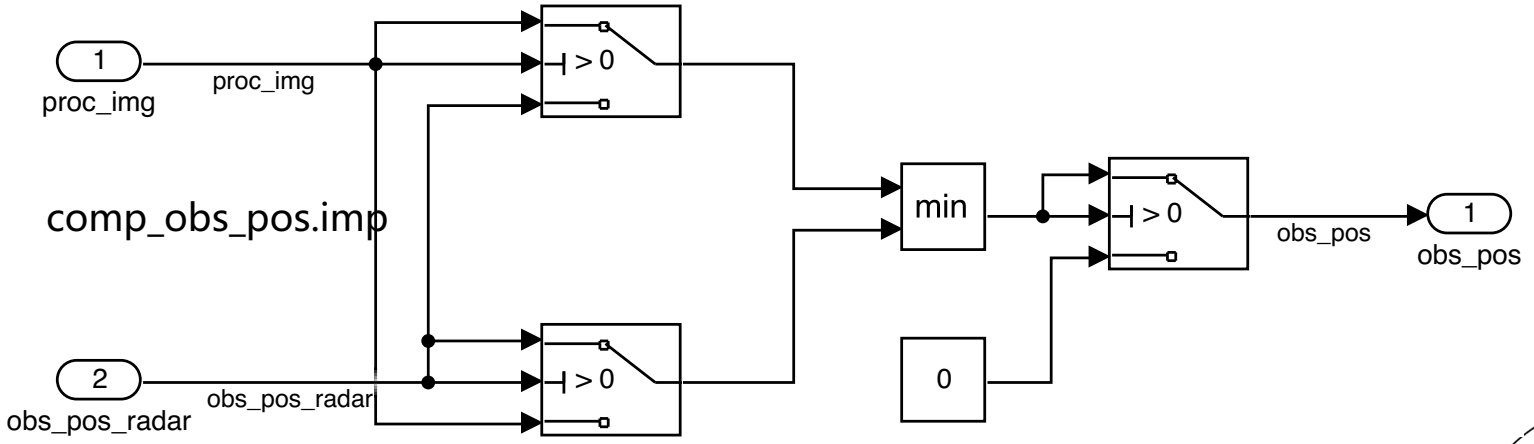
Software

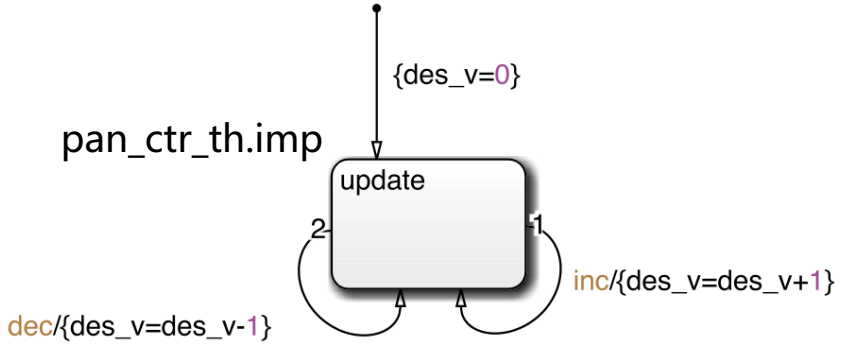
Platform

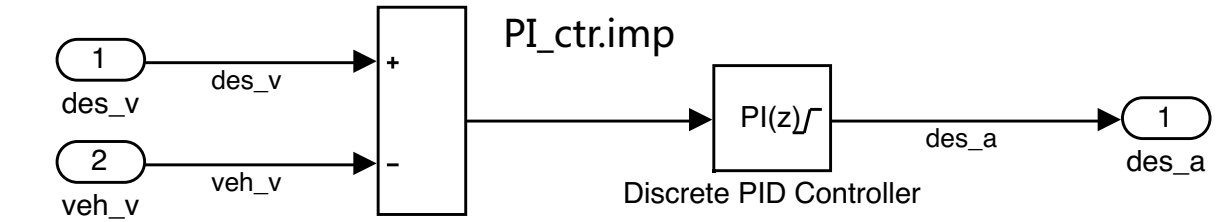


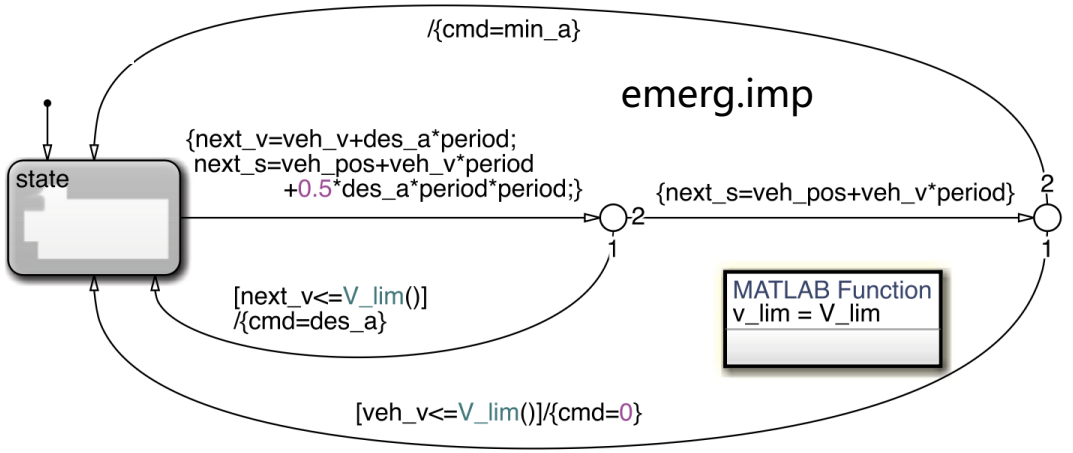

velo_voter.imp
2
wheel_v
wheel_v
1
wheel_valid
wheel_valid
4
laser_v
laser_v
3
laser_valid
laser_valid
> 0
> 0
+
+
2
×
÷
veh_v
1
veh_v


img_acp.imp
1
img
image
1/z
processing
new_image
1
proc_img


1
proc_img
proc_img
> 0
comp_obs_pos.imp
min
> 0
obs_pos
1
obs_pos
2
obs_pos_radar
obs_pos_radar
> 0
0


{des_v=0}
pan_ctr_th.imp
update
2
1
inc/{des_v=des_v+1}
dec/{des_v=des_v-1}


PI_ctr.imp
1
des_v
des_v
2
veh_v
veh_v
+
-
PI(z)
Discrete PID Controller
des_a
1
des_a

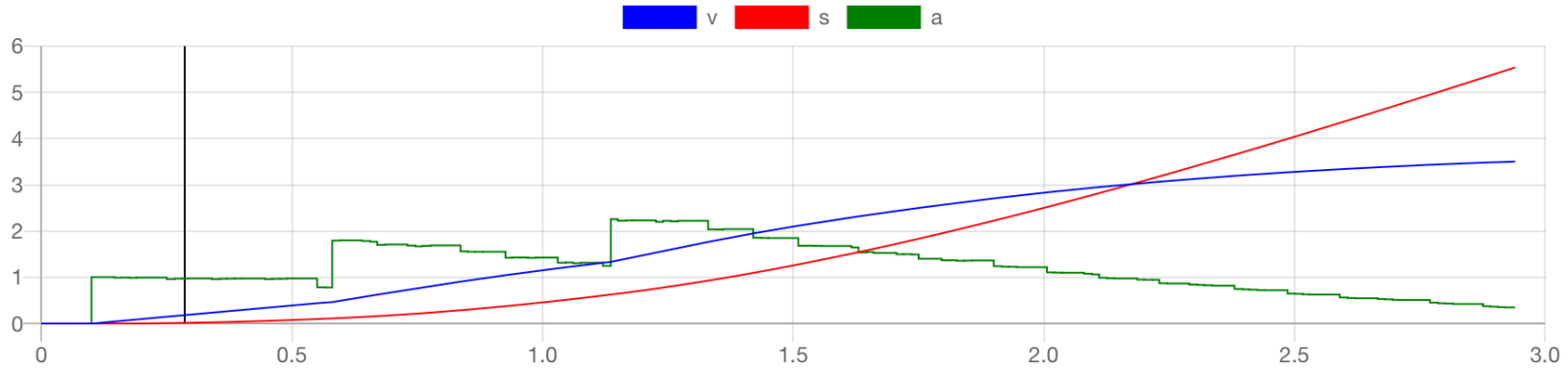

/{cmd=min_a}
emerg.imp
state
{next_v=veh_v+des_a*period;
next_s=veh_pos+veh_v*period
+0.5*des_a*period*period;}
{next_s=veh_pos+veh_v*period}
2
1
2
1
[next_v<=V_lim()]
/{cmd=des_a}
MATLAB Function
v_lim = V_lim
[veh_v<=V_lim()]/{cmd=0}

HCSP Simulator | Read HCSP File | system.txt | Number of steps: 100000 | Showing 10000 | starting from 0

10000+ events. Cur step | Show events only | Show details

Process: vehicle_imp Show Process Show callstack

```
v:  0.185
s:  0.017
PHY_vehicle_imp_veh_a:  0
PHY_vehicle_imp_veh_s:  1
PHY_vehicle_imp_veh_v_l:  1
PHY_vehicle_imp_veh_v_w:  1
a:  0.974   Hide graph
```



Process: ACT_img_acq_imp Hide Process Show callstack

```
(
  dis["img_acq_imp"]!;
  wait(0.045)
)**
   Show graph
```

Process: img_acq_imp Hide Process Show callstack

```
9856 step
9857 delay 0.001
9858 step
9859 step
9860 IO inputs[actuator][cmd] 0.974
9861 step
9862 IO outputs[PHY_vehicle_imp][veh_a] 0.974
9863 step
9864 IO block[emerg_imp]
9865 step
9866 IO free[0][emerg_imp]
9867 step
```

scheduler: Show Process Show callstack

vehicle_imp: Show Process Show callstack

ACT_img_acq_imp: Hide Process Show callstack

img_acq_imp: Hide Process Show callstack

ACT_comp_obs_pos_imp: Show Process Show callstack

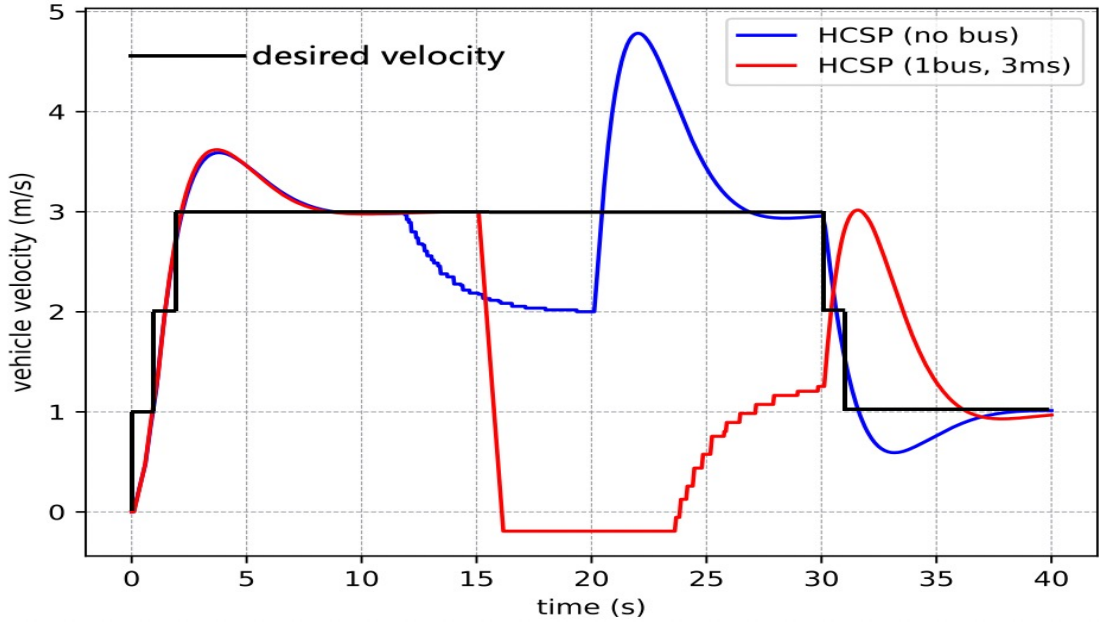
comp_obs_pos_imp: Show Process Show callstack

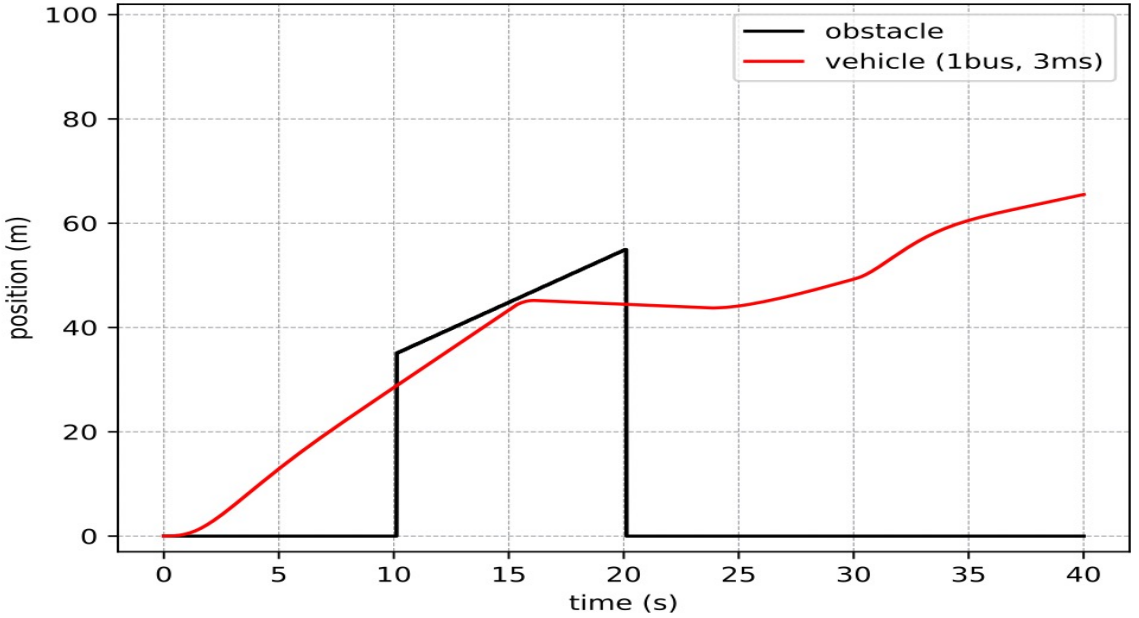
ACT_PI_ctr_imp: Show Process Show callstack

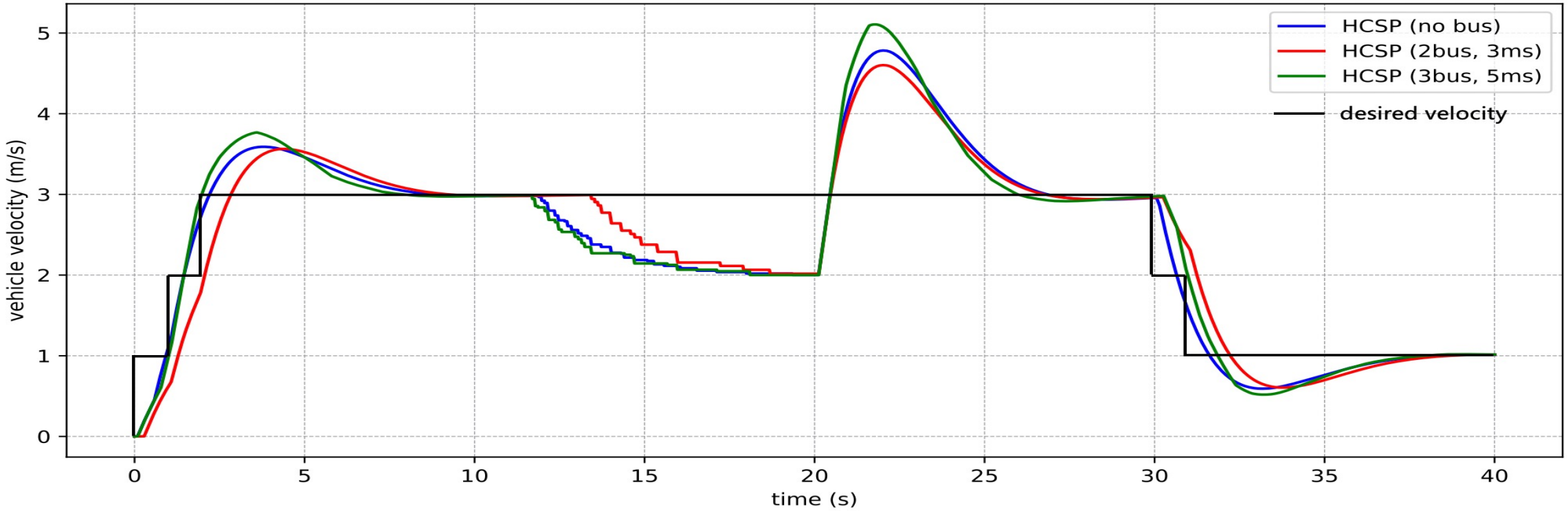
PI_ctr_imp: Hide Process Show callstack

Callstacks:

```
vehicle_imp:line18:      skip,
```


desired velocity
HCSP (no bus)
HCSP (1bus, 3ms)
vehicle velocity (m/s)
time (s)


obstacle
vehicle (1bus, 3ms)
position (m)
time (s)


HCSP (no bus)
HCSP (2bus, 3ms)
HCSP (3bus, 5ms)
desired velocity
vehicle velocity (m/s)
time (s)

Case study: a cruise control system (Verification)

Abstracted HCSP processes

$$
\begin{array}{lll}
\text{System} & ::= & \text{Plant} \| \text{Control} \\
\text{Plant} & ::= & v := v_0; s := s_0; P2C!v; P2C!s; C2P?a; (\langle \dot{s} = v, \dot{v} = a \& \mathbf{true} \rangle \trianglerighteq [](P2C!v \longrightarrow (P2C!s; C2P?a)))^* \\
\text{Control} & ::= & \left(P2C?v; P2C?s; v' := v + a_{des} \cdot period; s' := s + v \cdot period + \frac{1}{2} \cdot a_{des} \cdot period^2; C2P!a(s,v); \text{wait}(period)\right)^*
\end{array}
$$

$$
v_{lim}(s) = \begin{cases} v_{max}, & \text{if } s_{obs} - s \geq \frac{v_{max}^2}{-2a_{min}} \\ \sqrt{-2a_{min} \cdot (s_{obs} - s)}, & \text{if } 0 < s_{obs} - s < \frac{v_{max}^2}{-2a_{min}} \\ 0, & \text{otherwise} \end{cases}
$$

$$
a(s,v) = \begin{cases} a_{des} & \text{if } v_{next} \leq v_{lim}(s_{next}) \\ 0 & \text{else if } v \leq v_{lim}(s + v \cdot period) \\ a_{min} & \text{otherwise} \end{cases}
$$

The safety property can be implied by the loop invariant $loop\_inv: s \leq s_{obs} \wedge v \leq v_{lim}(s)$

We can prove that $loop\_inv(s,v) \rightarrow loop\_inv(s',v')$ where $\begin{cases} v' = v + a(s,v) \cdot period \\ s' = s + v \cdot period + 0.5 \cdot a(s,v) \cdot period^2 \end{cases}$

Conclusion

- ✓ A combination of AADL and S/S (AADL⊕S/S)
- ✓ An HCSP-based analysis and verification for AADL⊕S/S
- ✓ A simulation tool for AADL⊕S/S
- ✓ An application of the workflow to modelling and analysing the case study of an automatic cruise control system

Future works

- ☐ Expand the subset of AADL components under consideration
- ☐ Explore ways to verify complex HCSP models using HHL

[J] Xiong Xu et al. **Unified graphical co-modeling, analysis and verification of cyber-physical systems by combining AADL and Simulink/Stateflow**. Theor. Comput. Sci. 903: 1-25 (2022)

Thanks for your attention!