Model-Based Design

SE 3BB4

Ryszard Janicki

Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada

Design

Concepts: **design process:**
requirements to **models** to implementations

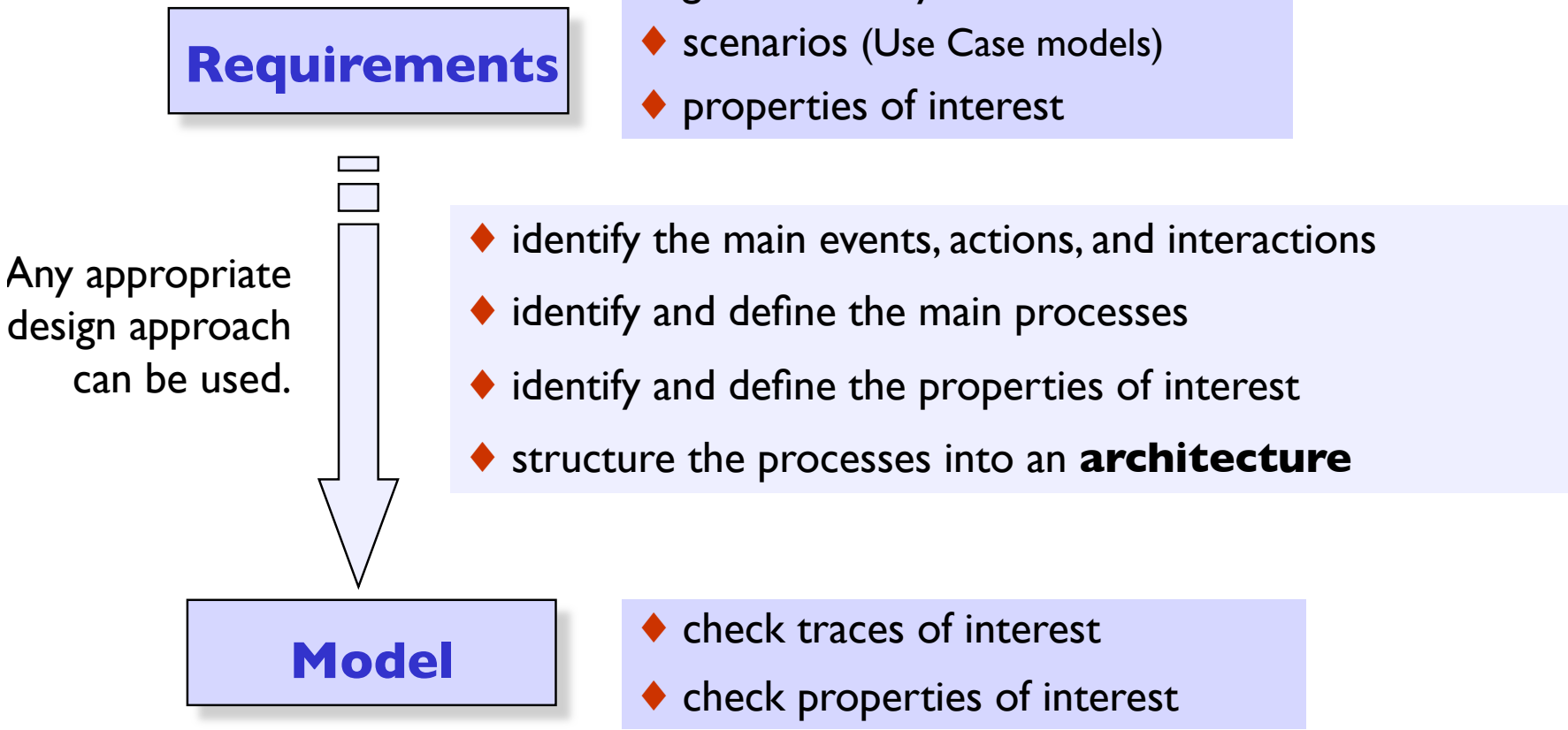
Models: check properties of interest:
- safety on the appropriate (sub)system
- progress on the overall system

Practice: model interpretation - to infer actual system behavior
threads and monitors

Aim: rigorous design process.

- The relationship between *Models* and *Requirements* is a cycle!

Requirements and Models



- An arrow from *Model* to *Requirements* is also needed!

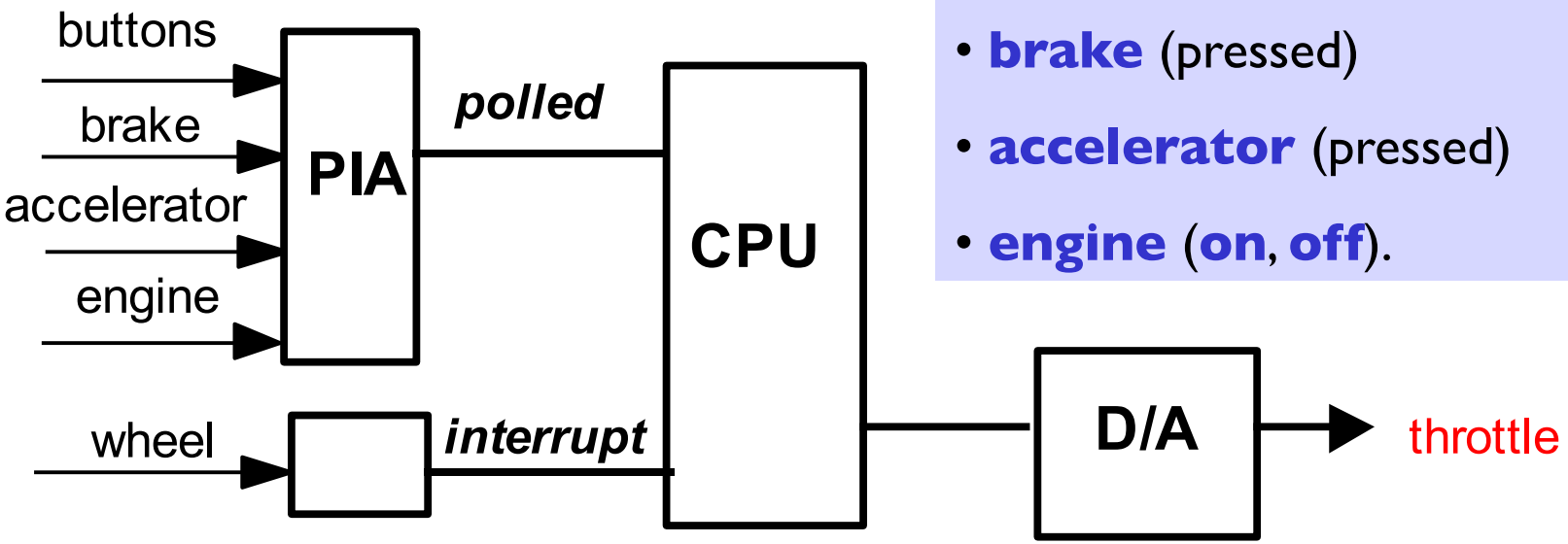
Cruise Control System: Requirements

- When the car ignition is switched on and the **on** button is pressed, the current speed is recorded and the system is enabled.
  It maintains the speed of the car at the recorded setting.
- Pressing the brake, accelerator or **off** button disables the system.
- Pressing **resume** or **on** enables the system.

Cruise Control System - Hardware

Parallel Interface Adapter (PIA) is polled every 100msec. It records the actions of the sensors:

- buttons (**on**, **off**, **resume**)
- **brake** (pressed)
- **accelerator** (pressed)
- **engine** (**on**, **off**).

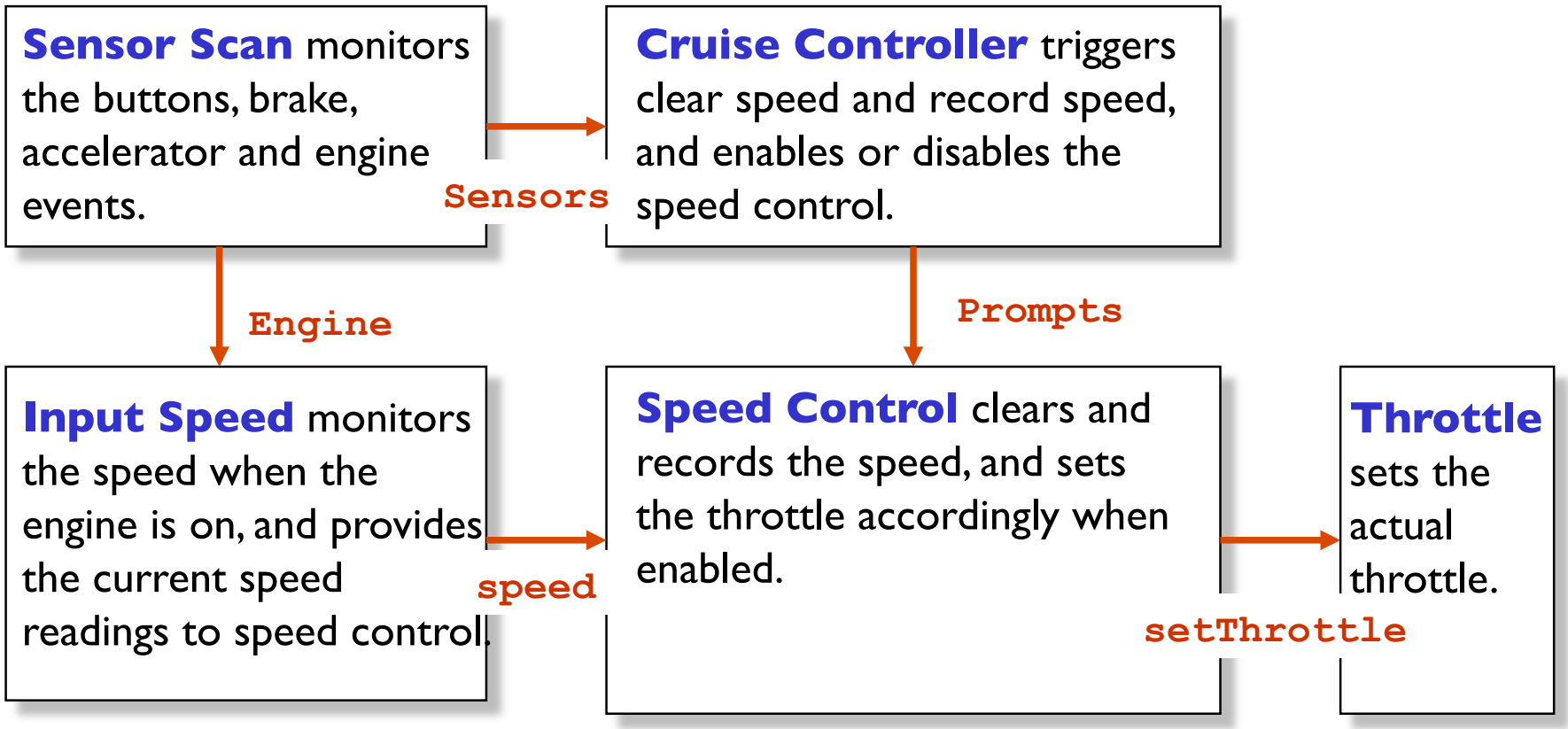


Wheel revolution sensor generates interrupts to enable the car **speed** to be calculated.

Output: The cruise control system controls the car speed by setting the **throttle** via the digital-to-analogue converter.

Outline Design

♦outline processes and interactions.



- *Gearbox* is not included, so it looks as an oversimplification!

- Main events, actions and interactions.

  `on`, `off`, `resume`, `brake`, `accelerator` } **Sensors**

  `engine on`, `engine off`,

  `speed`, `setThrottle`

  `clearSpeed`, `recordSpeed`, } **Prompts**

  `enableControl`, `disableControl`

- Identify main processes.

  `Sensor Scan`, `Input Speed`,

  `Cruise Controller`, `Speed Control` and

  `Throttle`

- Identify main properties.

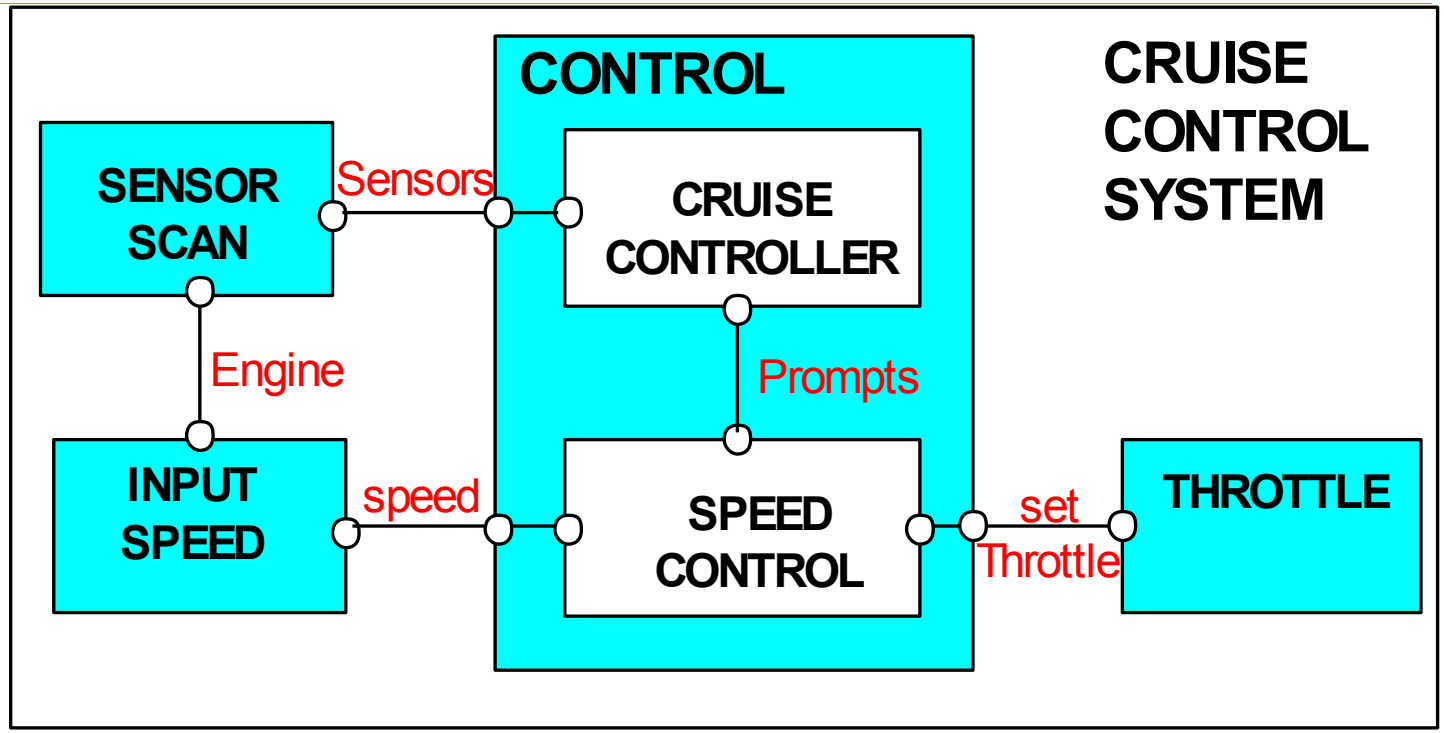
  safety - disabled when `off`, `brake` or `accelerator` pressed.

- Define and structure each process.

Structure, Actions and Interactions

The CONTROL system is structured as two processes.

The main actions and interactions are as shown.




```
set Sensors = {engineOn,engineOff,on,off,
               resume,brake,accelerator}
set Engine  = {engineOn,engineOff}
set Prompts = {clearSpeed,recordSpeed,
               enableControl,disableControl}
```

```
SENSORSCAN = ({Sensors} -> SENSORSCAN).
      // monitor speed when engine on
INPUTSPEED = (engineOn -> CHECKSPEED),
CHECKSPEED = (speed -> CHECKSPEED
              |engineOff -> INPUTSPEED
             ).
      // zoom when throttle set
THROTTLE =(setThrottle -> zoom -> THROTTLE).
      // perform speed control when enabled
SPEEDCONTROL = DISABLED,
DISABLED =({speed,clearSpeed,recordSpeed}->DISABLED
          | enableControl -> ENABLED
          ),
ENABLED = ( speed -> setThrottle -> ENABLED
          |{recordSpeed,enableControl} -> ENABLED
          | disableControl -> DISABLED
          ).
set DisableActions = {off,brake,accelerator}
      // enable speed control when cruising, disable when a disable action occurs
CRUISECONTROLLER = INACTIVE,
INACTIVE =(engineOn -> clearSpeed -> ACTIVE
          |DisableActions -> INACTIVE ),
ACTIVE   =(engineOff -> INACTIVE
          |on->recordSpeed->enableControl->CRUISING
          |DisableActions -> ACTIVE ),
CRUISING =(engineOff -> INACTIVE
          |DisableActions->disableControl->STANDBY
          |on->recordSpeed->enableControl->CRUISING ),
STANDBY  =(engineOff -> INACTIVE
          |resume -> enableControl -> CRUISING
          |on->recordSpeed->enableControl->CRUISING
          |DisableActions -> STANDBY
          ).
```

$\|\ CONTROL = (CRUISECONTROLLER \|\ SPEEDCONTROL).$

Some Problem

- $Cruising = (engineOff \rightarrow INACTIVE \parallel \ldots$
- WHY?
- Shouldn't be
  $Cruising = (engineOff \rightarrow disableCntrol \rightarrow INACTIVE \parallel \ldots$
  ?.

$\| \ CONTROL = (CRUISECONTROLLER \| SPEEDCONTROL).$

- The solutions, even not fully complete is complex. It is difficult, if not impossible, to deduct somrthing form it without using some tools!
  Which bothers me a lot!
- **Animation**
  Animation is a fishing expedition, but it may show some errors or unexpected behaviour. However showing no errors does not mean much!
- Here we can check the following traces:
  - Is control enabled after the engine is switched on and the on button is pressed?
  - Is control disabled when the brake is then pressed?
  - Is control re-enabled when resume when resume is pressed?
- However, we need analysis to check exhaustively:

Safety: Is the control disabled when **off**, **brake** or **accelerator** is pressed?

Progress: Can every action eventually be selected?

Safety Properties

- Safety checks are compositional. If there is no violation at a subsystem level, then there cannot be a violation when the subsystem is composed with other subsystems.
- This is because, if the ERROR state of a particular safety property is unreachable in the LTS of the subsystem, it remains unreachable in any subsequent parallel composition which includes the subsystem. Hence...
- Safety properties should be composed with the appropriate system or subsystem to which the property refers. In order that the property can check the actions in its alphabet, these actions must not be hidden in the system.

Safety properties

```
property CRUISESAFETY =
   ({DisableActions,disableControl} -> CRUISESAFETY
   |{on,resume} -> SAFETYCHECK
   ),
SAFETYCHECK =
   ({on,resume} -> SAFETYCHECK
   |DisableActions -> SAFETYACTION
   |disableControl -> CRUISESAFETY
   ),
SAFETYACTION =(disableControl->CRUISESAFETY).
```

LTS?

```
||CONTROL =(CRUISECONTROLLER
                ||SPEEDCONTROL
                ||CRUISESAFETY
               ).
```

Is CRUISESAFETY violated?

engineOn $\rightarrow$ *clearSpeed* $\rightarrow$ *on* $\rightarrow$ *recordSpeed* $\rightarrow$ *enableControl* $\rightarrow$ *engineOff* $\rightarrow$ *off* $\rightarrow$ *off*

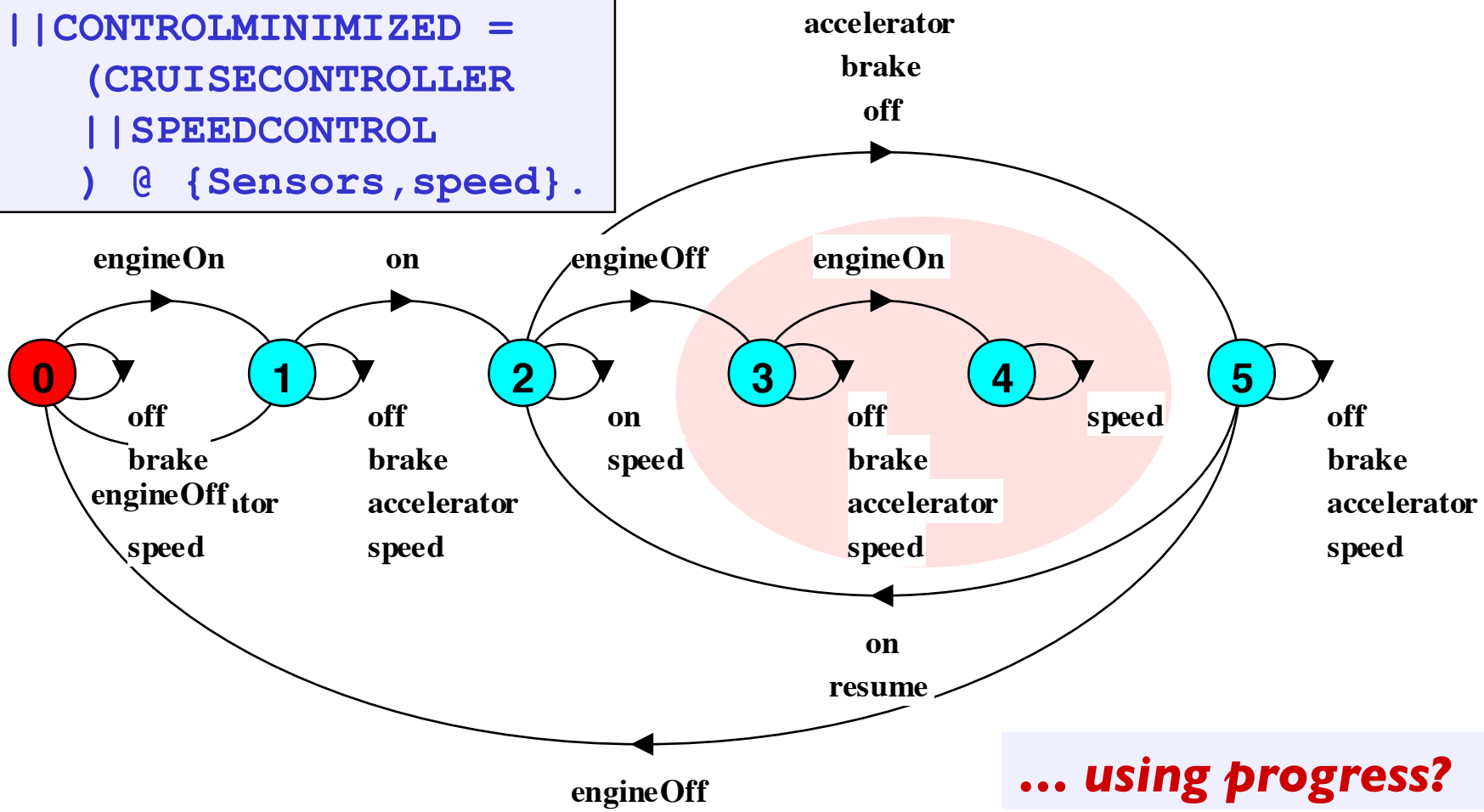
- CRUISESAFETY violation!

$engineOn \rightarrow clearSpeed \rightarrow on \rightarrow recordSpeed \rightarrow enableControl \rightarrow engineOff \rightarrow off \rightarrow off$

- CRUISESAFETY violation! **Strange Circumstances!**
  If the system is enabled by switching the engine on and pressing the on button, and then the engine is switched off, it appears that the control system is not disabled.
- What if the engine is switched on again?
- Consider the trace:
  $engineOn \rightarrow clearSpeed \rightarrow on \rightarrow recordSpeed \rightarrow enableControl \rightarrow engineOff \rightarrow engineOn \rightarrow speed \rightarrow setThrottle \rightarrow speed \rightarrow setThrottle \rightarrow \ldots$
- The car will accelerate and zoom off when the engine is switched on again!

- Action hiding and minimization can help to reduce the size of an LTS diagram and make it easier to interpret.

```
minimal
||CONTROLMINIMIZED =
    (CRUISECONTROLLER
    ||SPEEDCONTROL
    ) @ {Sensors,speed}.
```



... using progress?

Progress

- Progress violation for actions:
  $\{accelerator, brake, clearSpeed, disableControl, enableControl, engineOff, engineOn, off, on, recordSpeed, resume\}$
- Trace to terminal set of states:
  $engineOn \rightarrow clearSpeed \rightarrow on \rightarrow recordSpeed \rightarrow enableControl \rightarrow engineOff \rightarrow engineOn$
- Cycle in terminal set: $speed \rightleftarrows setThrottle$
- Actions in terminal set: $\{setThrottle, speed\}$
- The same problem is no safety property process is added and no hidden actions.

Revised Cruise Controller

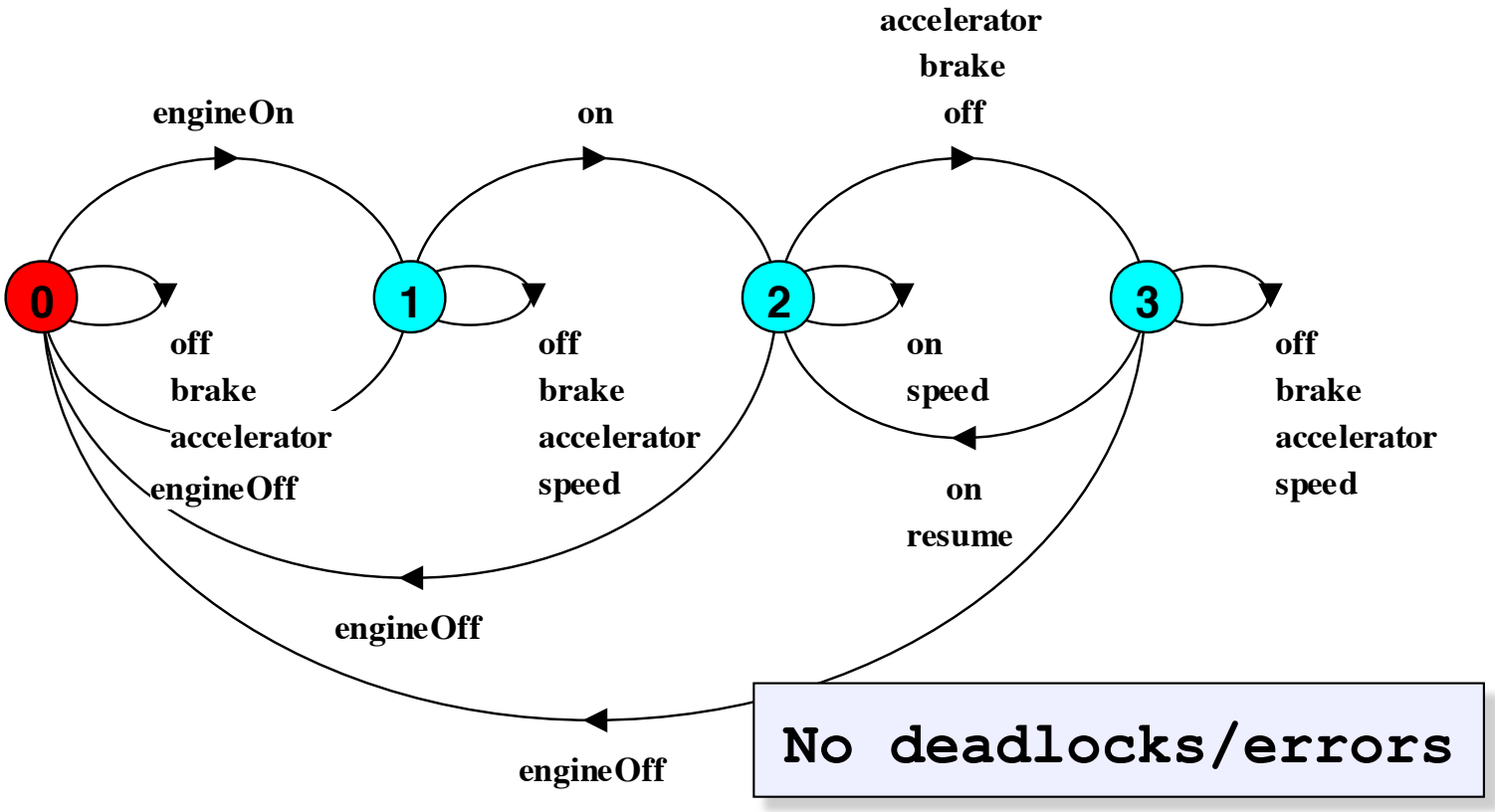
Modify **CRUISECONTROLLER** so that control is disabled when the engine is switched off:

```
...
CRUISING =(engineOff -> disableControl -> INACTIVE
          |DisableActions -> disableControl -> STANDBY
          |on->recordSpeed->enableControl->CRUISING
          ),
...
```

Modify the safety property:

```
property IMPROVEDSAFETY =
  {DisableActions,disableControl,engineOff} -> IMPROVEDSAFETY
  |{on,resume}   -> SAFETYCHECK
  ),
SAFETYCHECK = ({on,resume}  -> SAFETYCHECK
              |{DisableActions,engineOff} -> SAFETYACTION
              |disableControl     -> IMPROVEDSAFETY
              ),
SAFETYACTION =(disableControl     -> IMPROVEDSAFETY).
```

OK now?



We can now proceed to compose the whole system:

```
||CONTROL =
   (CRUISECONTROLLER||SPEEDCONTROL||CRUISESAFETY
   )@ {Sensors,speed,setThrottle}.
||CRUISECONTROLSYSTEM =
     (CONTROL||SENSORSCAN||INPUTSPEED||THROTTLE).
```

- No deadlocks/errors

Progress Properties

- Progress checks are not compositional.
- Even if there is no violation at a subsystem level, there may still be a violation when the subsystem is composed with other subsystems.
- This is because an action in the subsystem may satisfy progress yet be unreachable when the subsystem is composed with other subsystems which constrain its behaviour. Hence...
- Progress checks should be conducted on the complete target system after satisfactory completion of the safety checks.
- No *progress* violation for the solution from previous page.

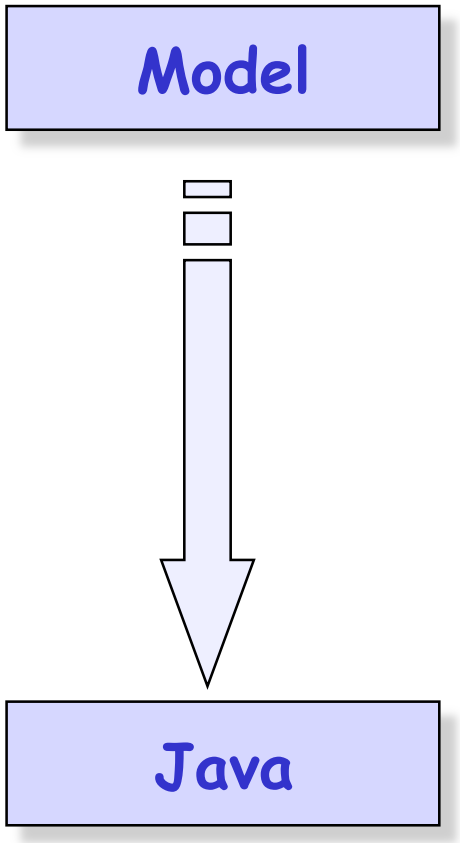
System Sensitivities

- What about progress under adverse conditions?
- Check for system sensitivities.
  $\| SPEEDHIGH = CRUISECONTROLSYSTEM << \{speed\}$.
- Progress violation for actions:
  $\{accelerator, brake, engineOff, engineOn, off, on, resume, setThrottle, zoom\}$
- Path to terminal set of states: $engineOn \rightarrow tau$
- Cycle in terminal set: $speed$ Actions in terminal set: $\{speed\}$
- The system may be sensitive to the priority of the action *speed*.

Model Interpretation

- Models can be used to indicate system sensitivities.
- If it is possible that erroneous situations detected in the model may occur in the implemented system, then the model should be revised to find a design which ensures that those violations are avoided.
- However, if it is considered that the real system will not exhibit this behavior, then no further model revisions are necessary.
- Model interpretation and correspondence to the implementation are important in determining the relevance and adequacy of the model design and its analysis.

From Models to Implementations: One Possibility



- identify the main active entities
  - to be implemented as threads
- identify the main (shared) passive entities
  - to be implemented as monitors
- identify the interactive display environment
  - to be implemented as associated classes
- structure the classes as a class diagram

Cruise Control Again. Petri Nets

- **PROBLEM**
- Consider *Cruise Control System* discussed in this Lecture Notes and in Chapter 8 of the textbook.
- Provide a Petri Net model of it. You can any kind of Petri Nets discussed in class, i.e. elementary nets, predicate/transitions nets or coloured nets.
- Provide a few invariants as defined in Lecture Notes 12 (elementary nets are also place/transition nets)
- Prove lack of deadlock (or other property) as in Lecture Notes 12.

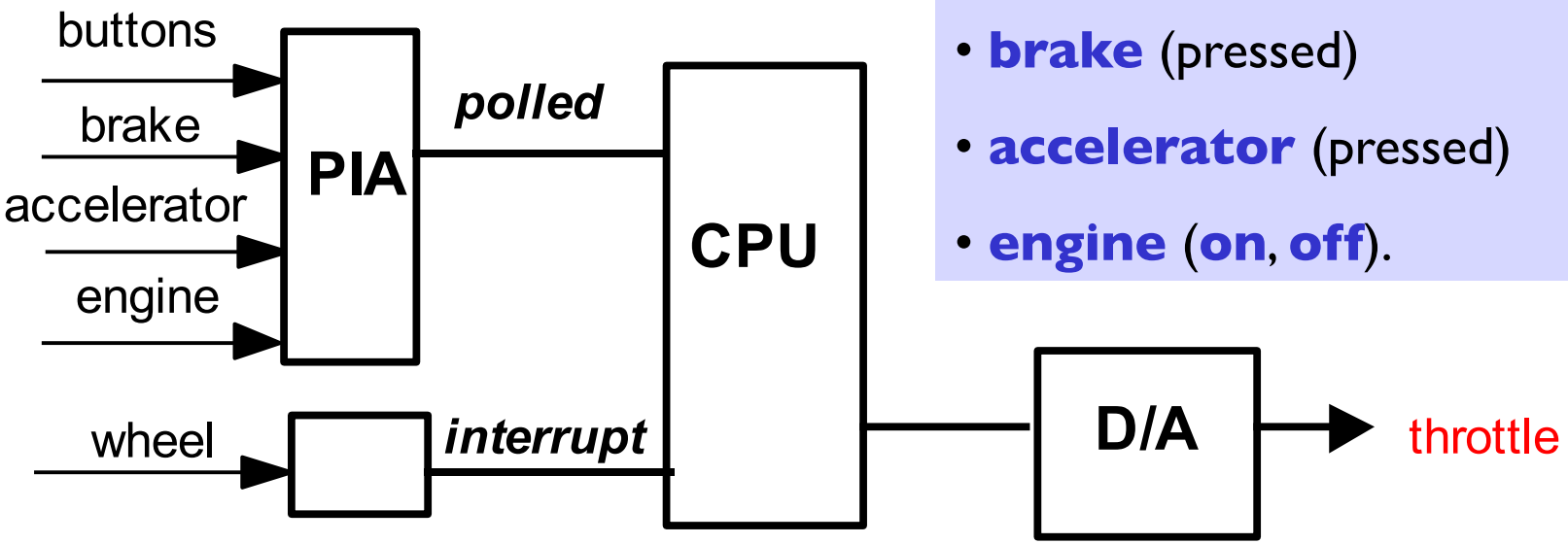
Cruise Control System: Requirements

- When the car ignition is switched on and the **on** button is pressed, the current speed is recorded and the system is enabled.
  It maintains the speed of the car at the recorded setting.
- Pressing the brake, accelerator or **off** button disables the system.
- Pressing **resume** or **on** enables the system.

Cruise Control System - Hardware

Parallel Interface Adapter (PIA) is polled every 100msec. It records the actions of the sensors:

- buttons (**on**, **off**, **resume**)
- **brake** (pressed)
- **accelerator** (pressed)
- **engine** (**on**, **off**).



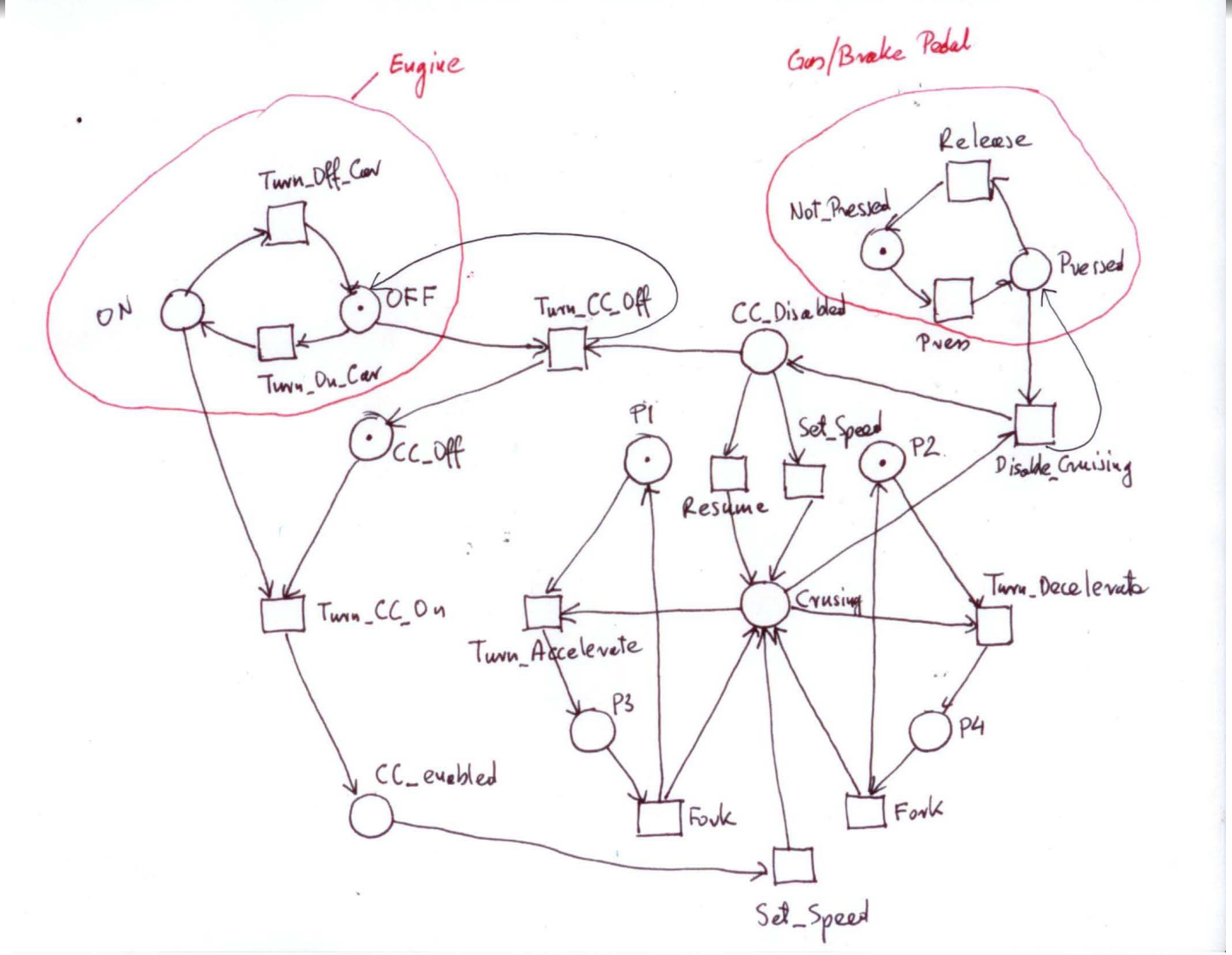
Wheel revolution sensor generates interrupts to enable the car **speed** to be calculated.

Output: The cruise control system controls the car speed by setting the **throttle** via the digital-to-analogue converter.

Cruise Control System: Petri Net Assumption

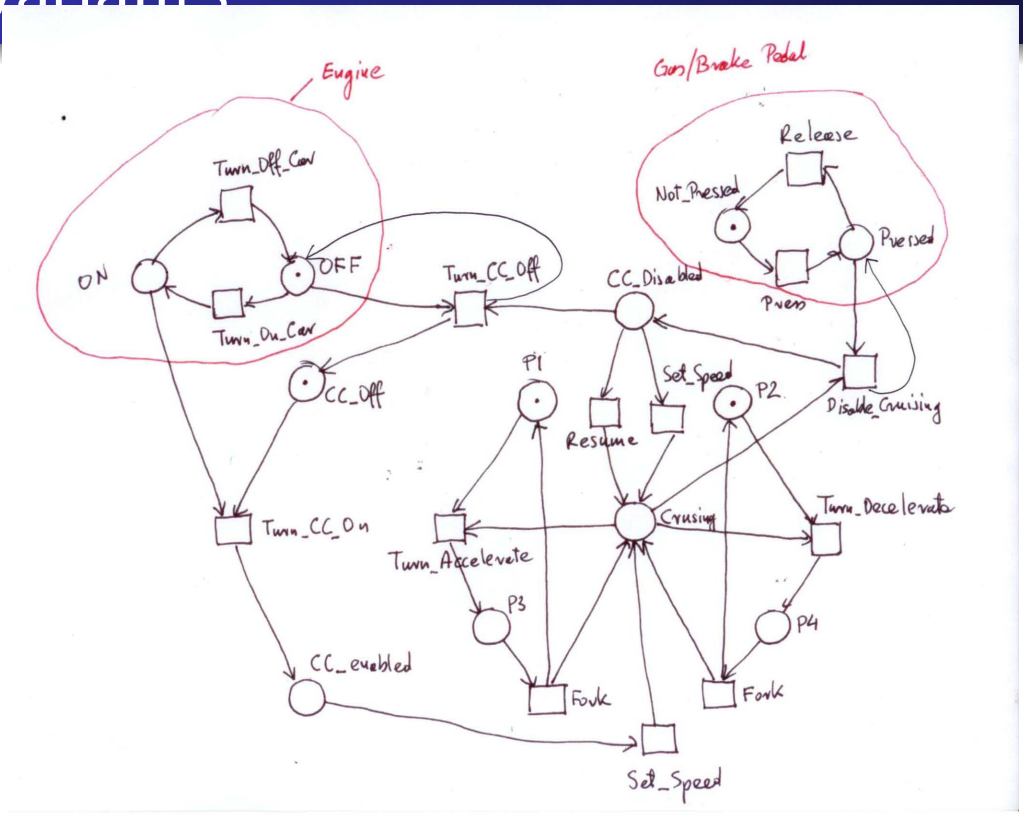
This solution uses Elementary Petri nets.

- The six Cruise Control (CC) actions are implemented as pairs of actions on three separate steering column turn levels: *on/off*, *set_speed/resume_speed*, *and accelerate/decelerate*.
- Order of precedence for the CC actions:
  1. *on/off*
  2. *set_speed/resume_speed*
  3. *accelerate/decelerate*
- When the cruise control system is enabled and either the accelerator pedal or brake pedal is depressed, the cruise control system is disabled and remains so until either *set_speed* or *resume_speed* is activated.
- Cruising speed in uphill and downhill driving conditions is not maintained, and thus not modelled in the following specification.



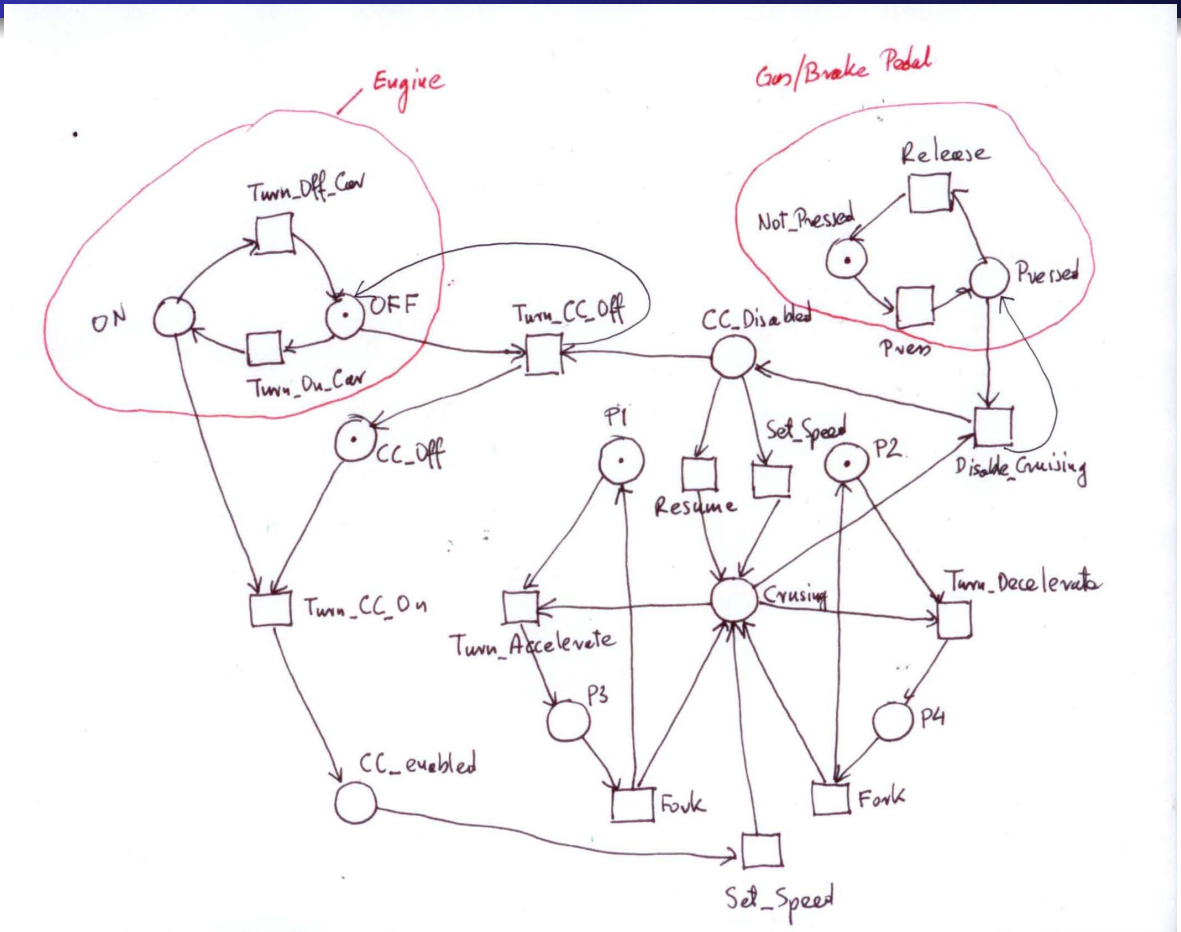
Gas pedal and brake pedal are two different devices but they are both used by the same right leg and are not expected to be used simultaneously. Hence they are modelled by the same subnet. If engine if off, pressing gas/brake does not affect Cruise Control (CC), so we have transitions *Press* and *Press_CC*. The rest of the net is rather self-explaining.

Invariants



- $m(ON) + m(OFF) = 1$
- $m(Not\_Pressed) + m(Pressed) = 1$
- $m(CC\_Off) + m(CC\_enabled) + m(Cruising) + m(P3) = 1$
- $m(CC\_Off) + m(CC\_enabled) + m(Cruising) + m(P4) = 1$
- $m(P1) + m(P3) = 1$
- $m(P2) + m(P4) = 1$

Deadlock



- From the invariant $m(ON) + m(OFF) = 1$ we have that either *Start_Car* or *Turn_Off_Car* is always enabled.
- Usually proving deadlock is much easier for Petri Nets than other models.