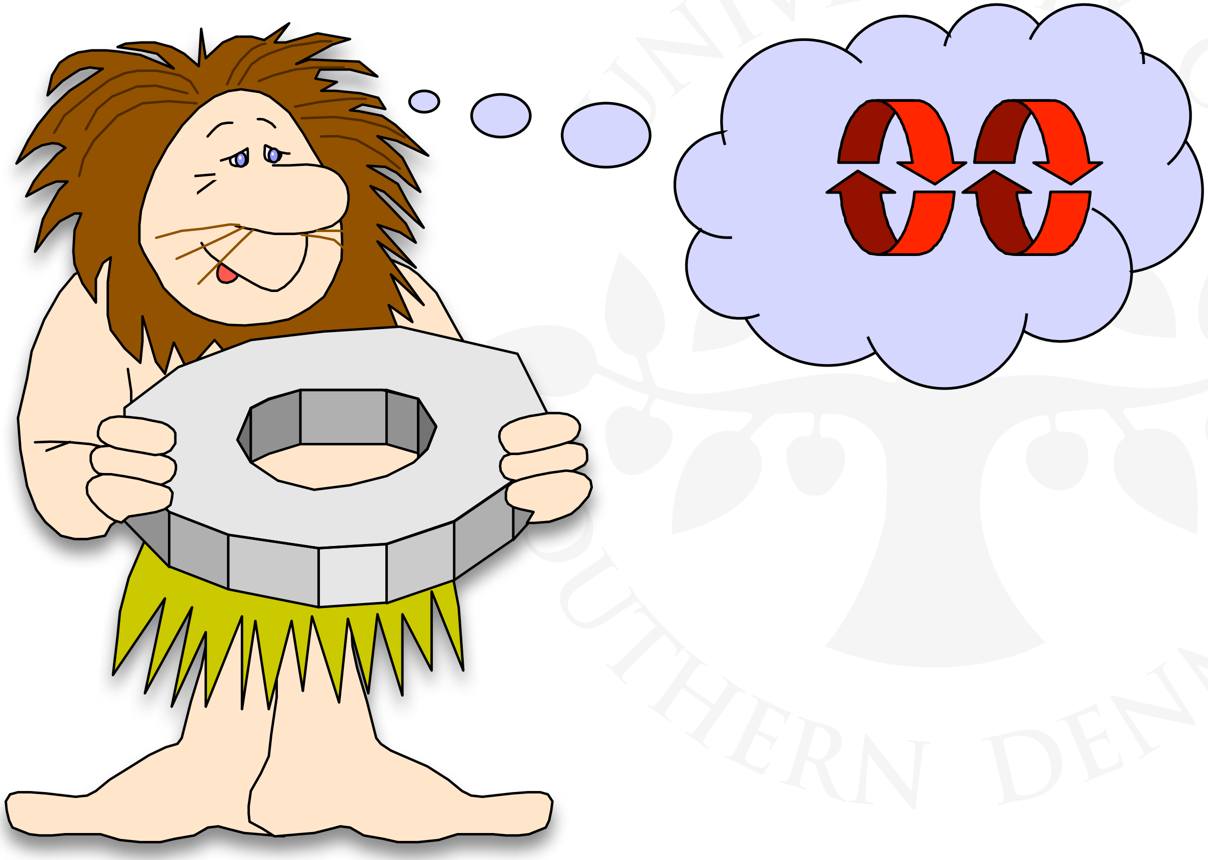
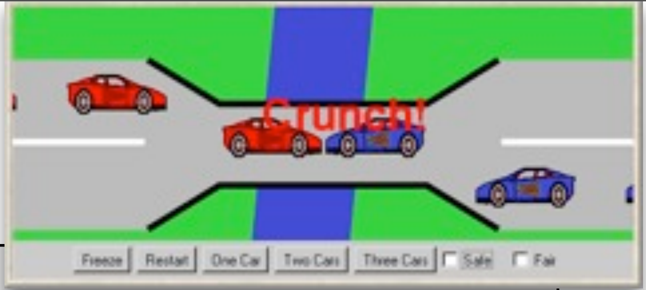
Chapter 8

Model-Based Design



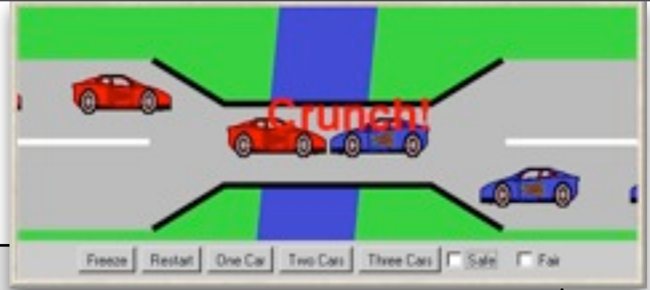
Repetition: Chapter 7
Safety & Liveness



A **safety** property **asserts** that nothing **bad** happens.

```
property ONEWAY = EMPTY,
EMPTY =     (red[ID].enter   -> RED[1]
            |blue[ID].enter -> BLUE[1]),

RED[i:ID] = (            red[ID].enter  -> RED[i+1]
            |when (i==1) red[ID].exit   -> EMPTY
            |when (i>1)  red[ID].exit   -> RED[i-1]),

BLUE[j:ID]= (            blue[ID].enter -> BLUE[j+1]
            |when (j==1) blue[ID].exit  -> EMPTY
            |when (j>1)  blue[ID].exit  -> BLUE[j-1]).
```

Repetition: Chapter 7
Safety & Liveness



A safety property **asserts** that nothing bad happens.

```
property ONEWAY = EMPTY,
EMPTY =      (red[ID].enter  -> RED[1]
             |blue[ID].enter -> BLUE[1]),

RED[i:ID] = (            red[ID].enter  -> RED[i+1]
            |when (i==1) red[ID].exit   -> EMPTY
            |when (i>1)  red[ID].exit   -> RED[i-1]),

BLUE[j:ID]= (            blue[ID].enter -> BLUE[j+1]
            |when (j==1) blue[ID].exit  -> EMPTY
            |when (j>1)  blue[ID].exit  -> BLUE[j-1]).
```

A liveness property **asserts** that something good eventually happens.

```
progress BLUECROSS = {blue[ID].enter}
progress REDCROSS =  {red[ID].enter}
```

Model-Based Design

Model-Based Design

Concepts:

design process:

- from requirements to **models**
- from **models** to implementations

Model-Based Design

Concepts:

design process:

- from requirements to **models**
- from **models** to implementations

Models:

check properties of interest:

- safety of the appropriate (sub-)system
- progress of the overall system

Model-Based Design

University of Southern Denmark

Concepts:

design process:

- from requirements to **models**
- from **models** to implementations

Models:

check properties of interest:

- safety of the appropriate (sub-)system
- progress of the overall system

Practice:

model "interpretation":

- to infer actual system behavior

active threads and passive monitors

Model-Based Design

UNIVERSITY OF SOUTHERN DENMARK

Concepts:

design process:

- from requirements to **models**
- from **models** to implementations

Models:

check properties of interest:

- safety of the appropriate (sub-)system
- progress of the overall system

Practice:

model “interpretation”:

- to infer actual system behavior

active threads and passive monitors

Aim: rigorous design process.

Model-Based Design

Case Study: *cruise controller*

Concepts:

design process:

- from requirements to **models**
- from **models** to implementations

Models:

check properties of interest:

- safety of the appropriate (sub-)system
- progress of the overall system

Practice:

model "interpretation":

- to infer actual system behavior

active threads and passive monitors

Aim: rigorous design process.

From Requirements To Models

8.1 From Requirements To Models

8.1 From Requirements To Models

Requirements

8.1 From Requirements To Models

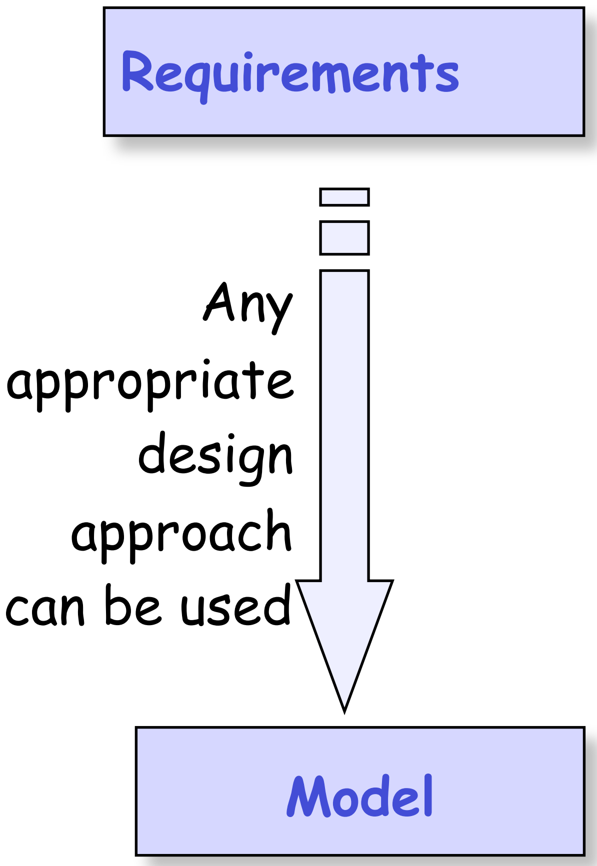
University of Southern Denmark

Requirements

- goals of the system
- scenarios (Use-Case models)
- properties of interest

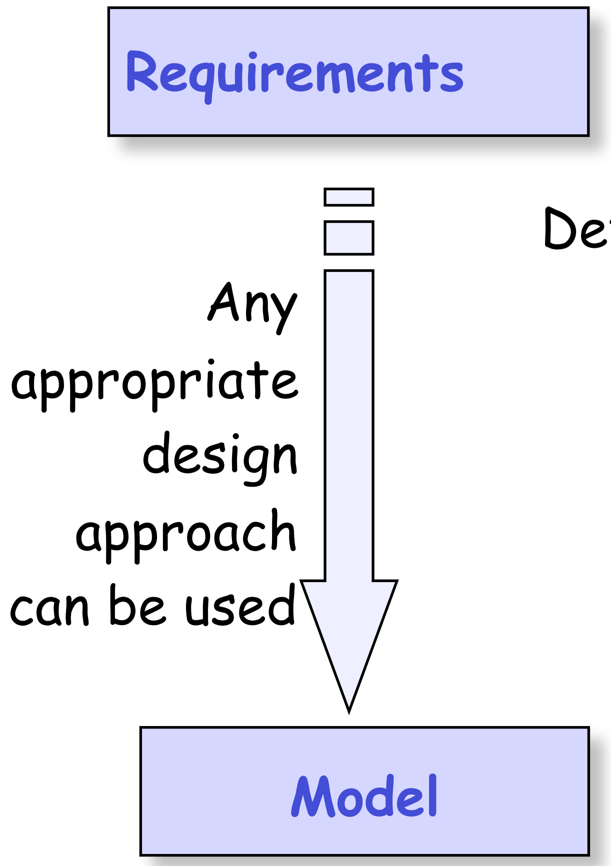
8.1 From Requirements To Models

- goals of the system
- scenarios (Use-Case models)
- properties of interest



8.1 From Requirements To Models

- goals of the system
- scenarios (Use-Case models)
- properties of interest



Determine appropriate level of abstraction

8.1 From Requirements To Models

- goals of the system
- scenarios (Use-Case models)
- properties of interest

Requirements

Any appropriate design approach can be used

Model

Determine appropriate level of abstraction

1. identify the main **events**, **actions**, and **interactions**

8.1 From Requirements To Models

University of Southern Denmark

Requirements

- goals of the system
- scenarios (Use-Case models)
- properties of interest

Any appropriate design approach can be used

Determine appropriate level of abstraction

1. identify the main **events**, **actions**, and **interactions**
2. identify and define the main **processes**

Model

8.1 From Requirements To Models

- goals of the system
- scenarios (Use-Case models)
- properties of interest

Requirements

Any appropriate design approach can be used

Model

Determine appropriate level of abstraction

1. identify the main **events**, **actions**, and **interactions**
2. identify and define the main **processes**
3. identify and define the **properties** of interest

8.1 From Requirements To Models

University of Southern Denmark

- goals of the system
- scenarios (Use-Case models)
- properties of interest

Requirements

Any appropriate design approach can be used

Model

Determine appropriate level of abstraction

1. identify the main **events**, **actions**, and **interactions**
2. identify and define the main **processes**
3. identify and define the **properties** of interest
4. structure the processes into an **architecture**

8.1 From Requirements To Models

- goals of the system
- scenarios (Use-Case models)
- properties of interest

Requirements

Any appropriate design approach can be used

Model

Determine appropriate level of abstraction

1. identify the main **events**, **actions**, and **interactions**
2. identify and define the main **processes**
3. identify and define the **properties** of interest
4. structure the processes into an **architecture**

- check traces of interest

8.1 From Requirements To Models

Requirements

- goals of the system
- scenarios (Use-Case models)
- properties of interest

Any appropriate design approach can be used

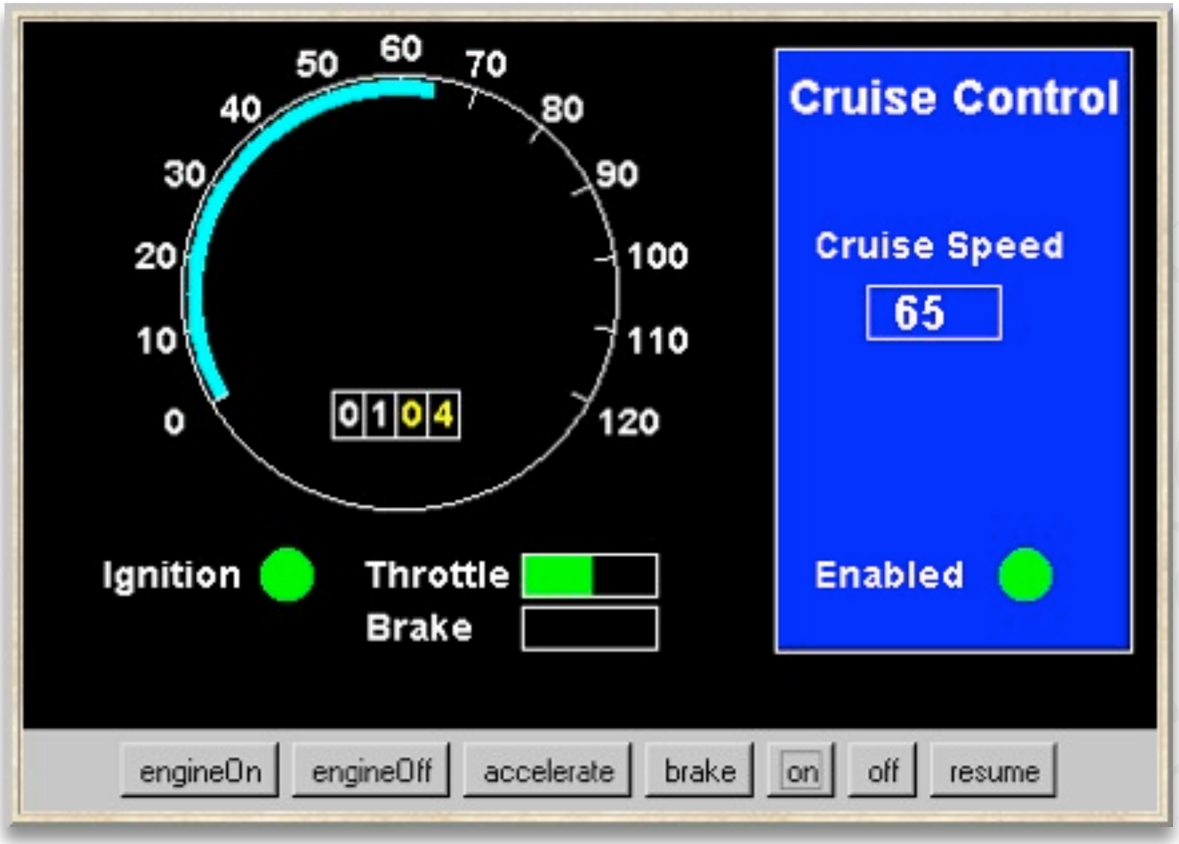
Determine appropriate level of abstraction

1. identify the main **events**, **actions**, and **interactions**
2. identify and define the main **processes**
3. identify and define the **properties** of interest
4. structure the processes into an **architecture**

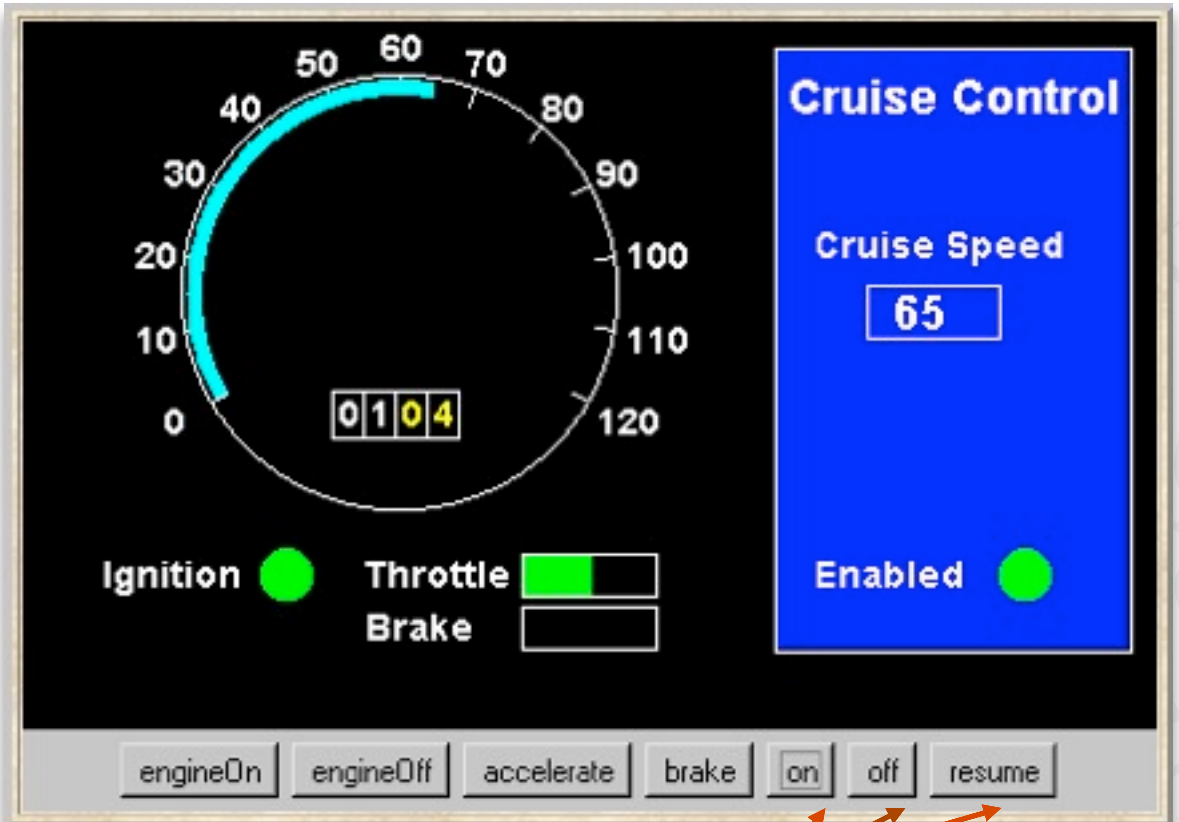
Model

- check traces of interest
- check properties of interest

Example: A Cruise Control System - Requirements

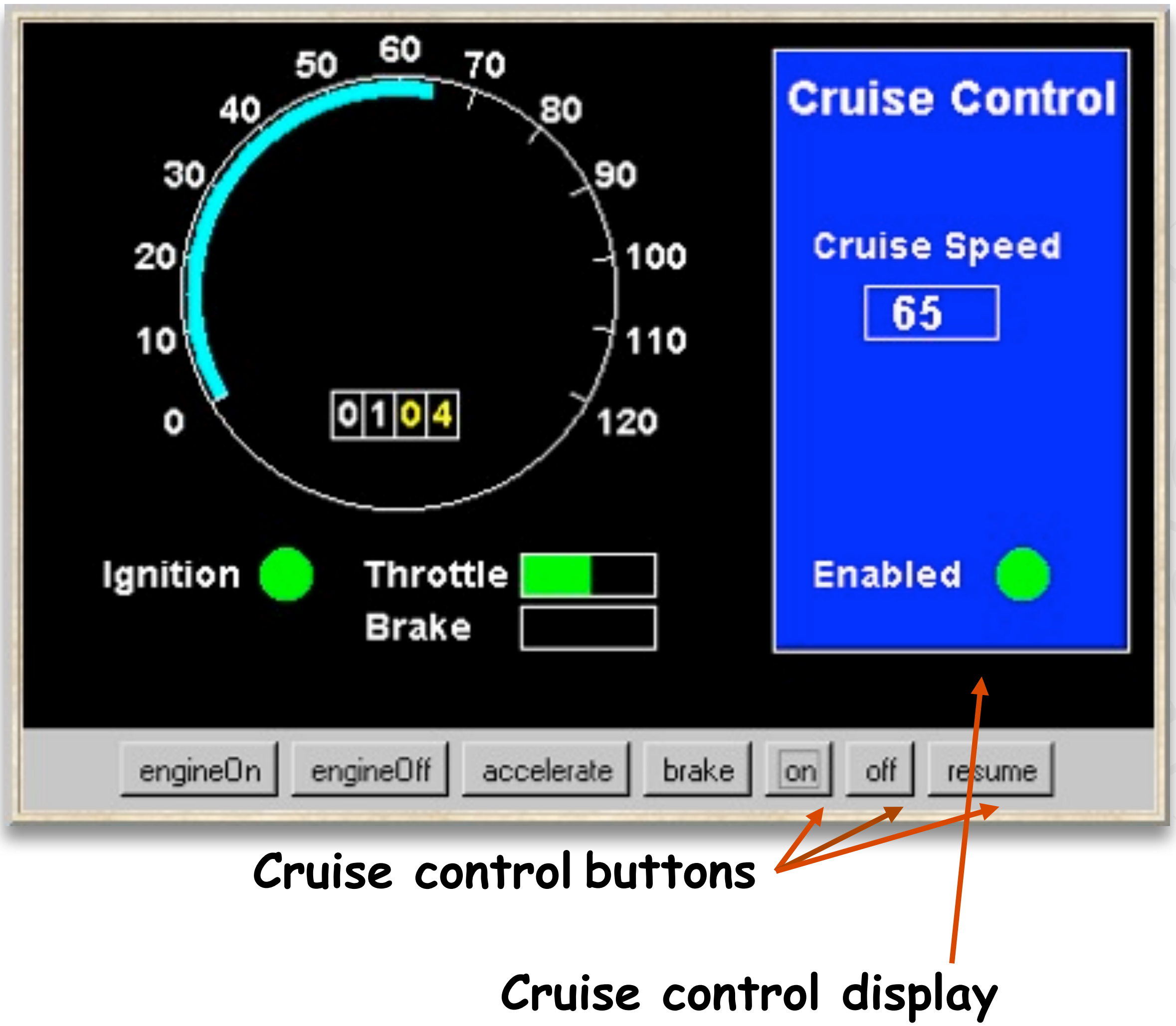


Example: A Cruise Control System - Requirements

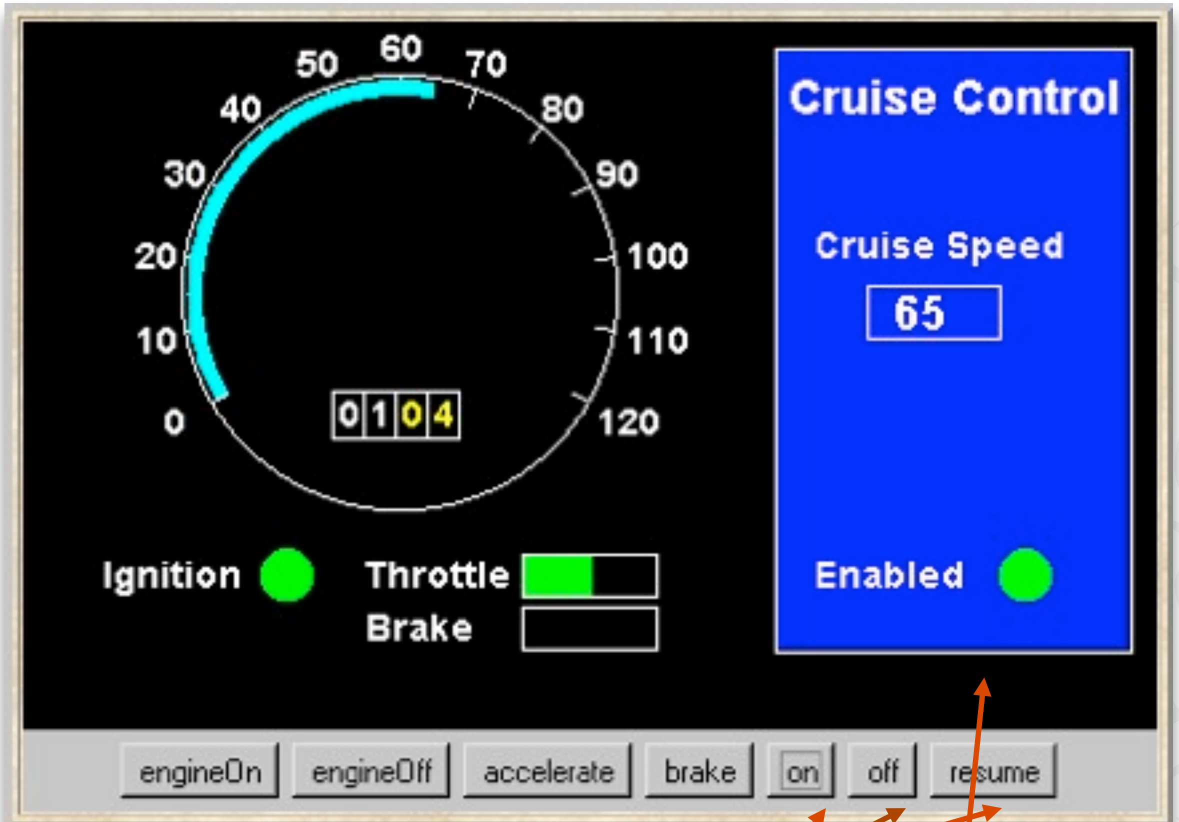


Cruise control buttons

Example: A Cruise Control System - Requirements



Example: A Cruise Control System - Requirements

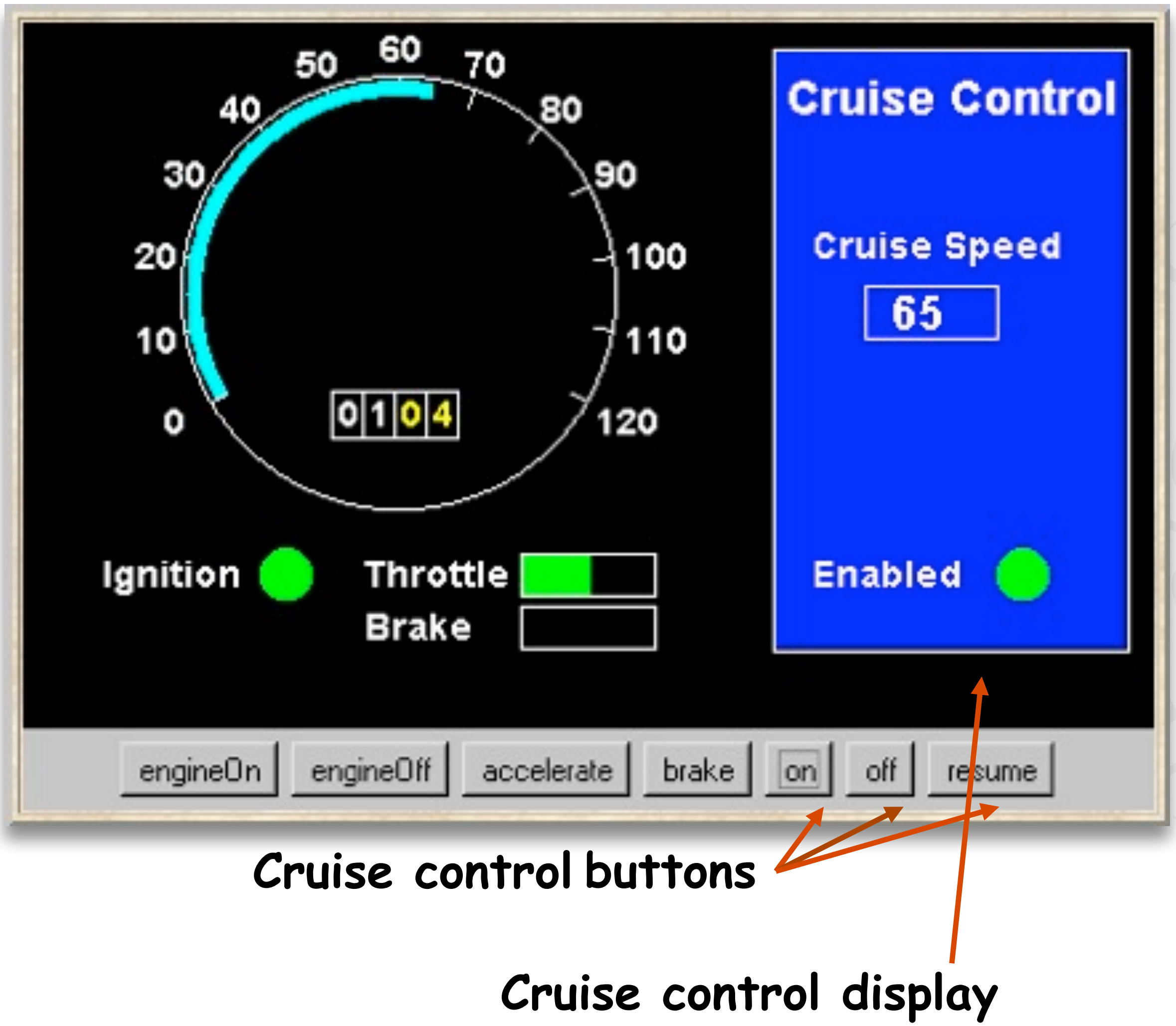


Cruise control buttons

Cruise control display

Requirements:

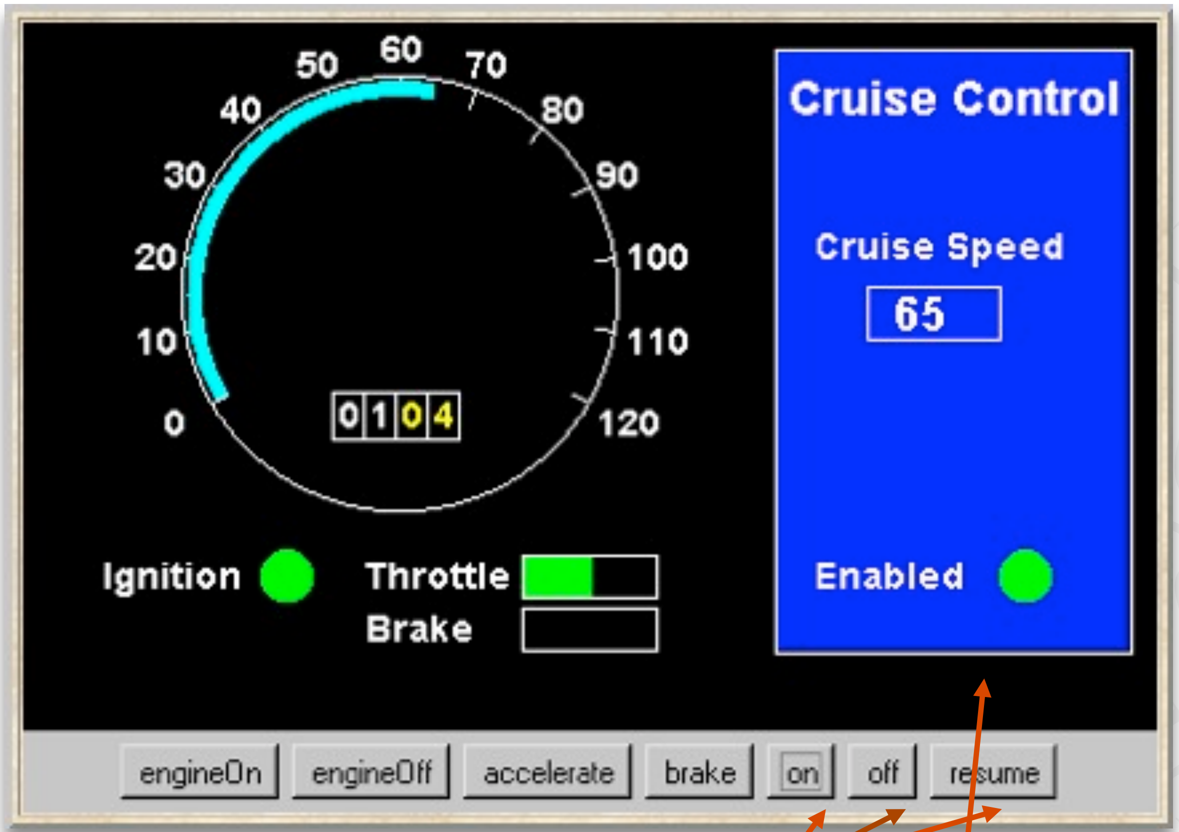
Example: A Cruise Control System - Requirements



Requirements:

Press '**on**' when the engine is started -> record current speed and enable the system;

Example: A Cruise Control System - Requirements



Cruise control buttons

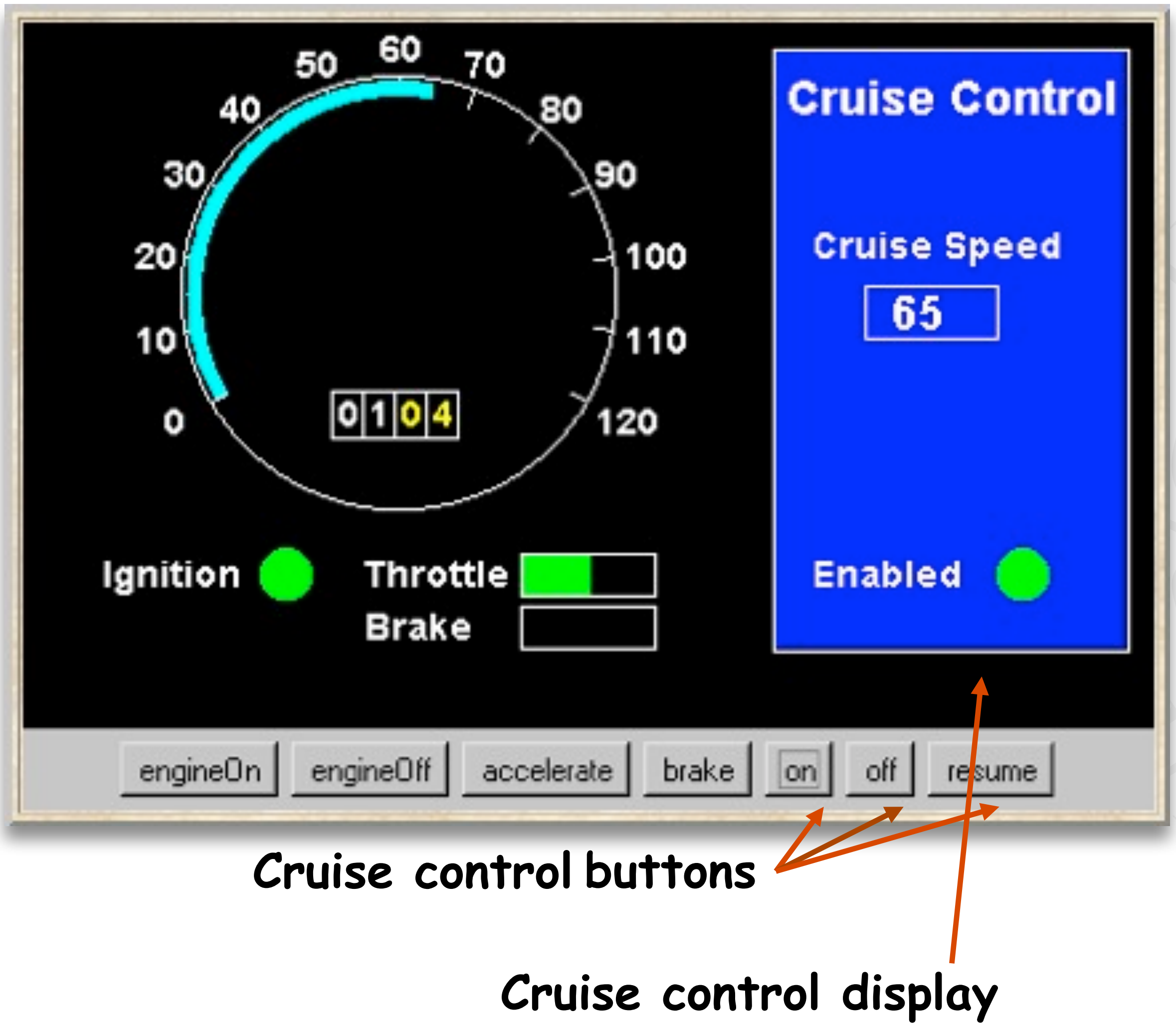
Cruise control display

Requirements:

Press 'on' when the engine is started -> record current speed and enable the system;

Maintain the recorded speed.

Example: A Cruise Control System - Requirements



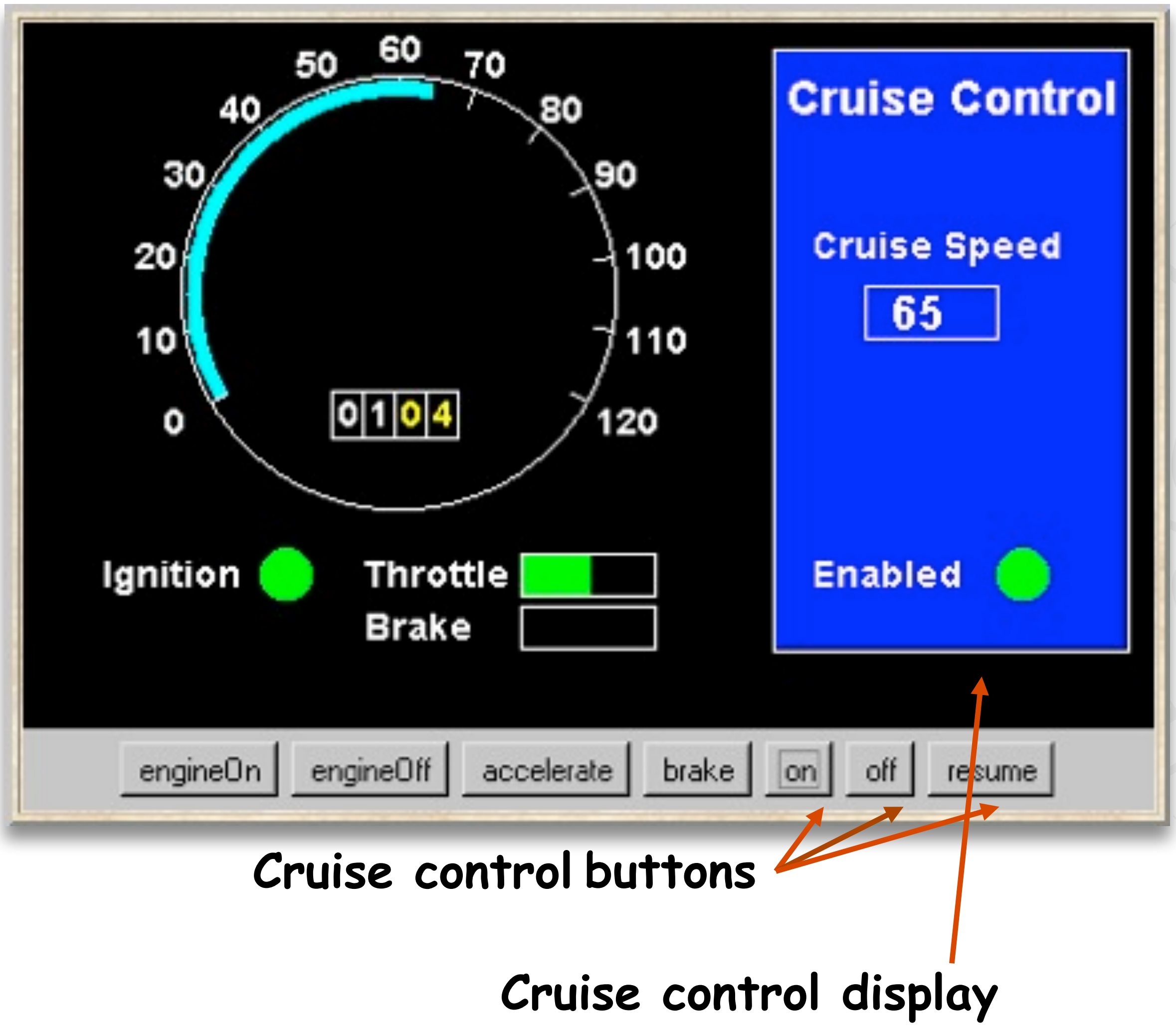
Requirements:

Press 'on' when the engine is started -> record current speed and enable the system;

Maintain the recorded speed.

'brake', 'accelerator' or 'off' -> disables the system.

Example: A Cruise Control System - Requirements



Requirements:

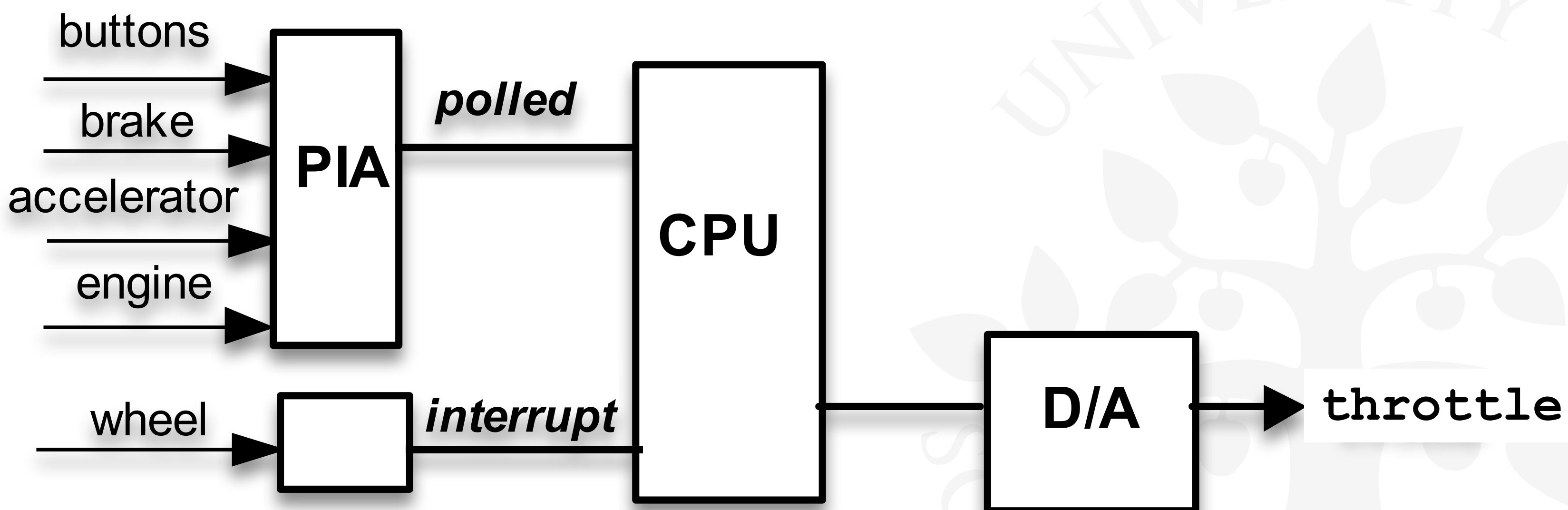
Press 'on' when the engine is started -> record current speed and enable the system;

Maintain the recorded speed.

'brake', 'accelerator' or 'off' -> disables the system.

'resume' or 'on' -> re-enables the system.

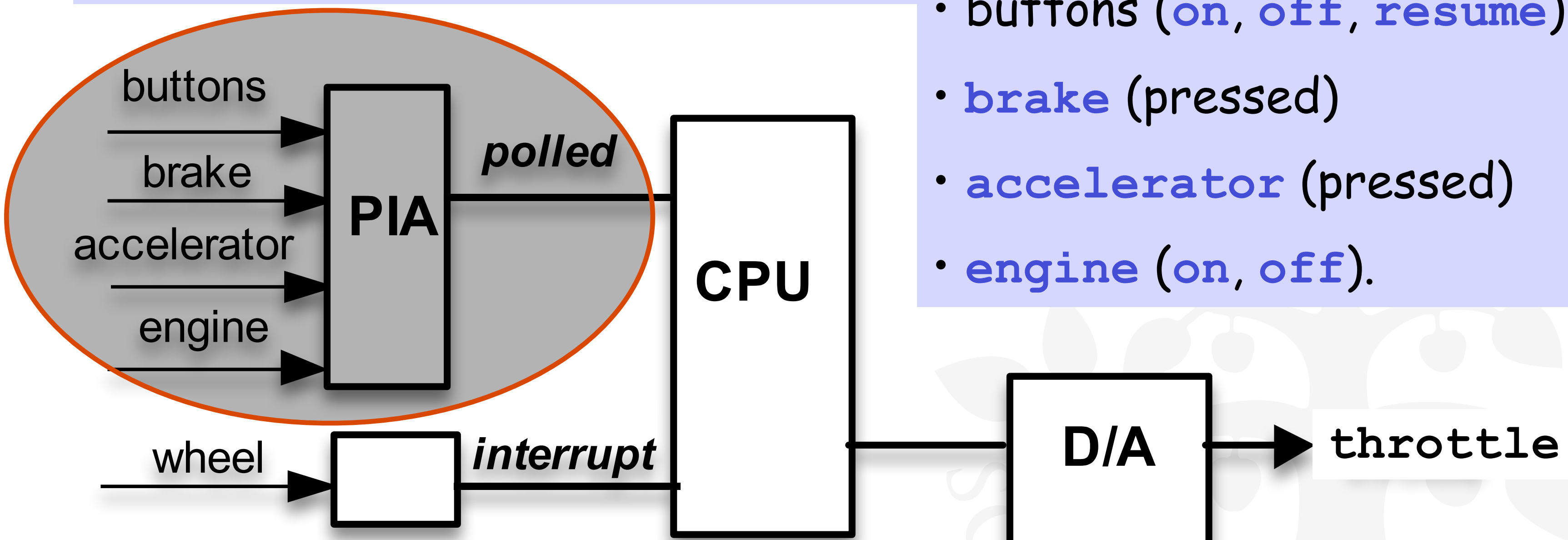
A Cruise Control System - Hardware



A Cruise Control System - Hardware

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

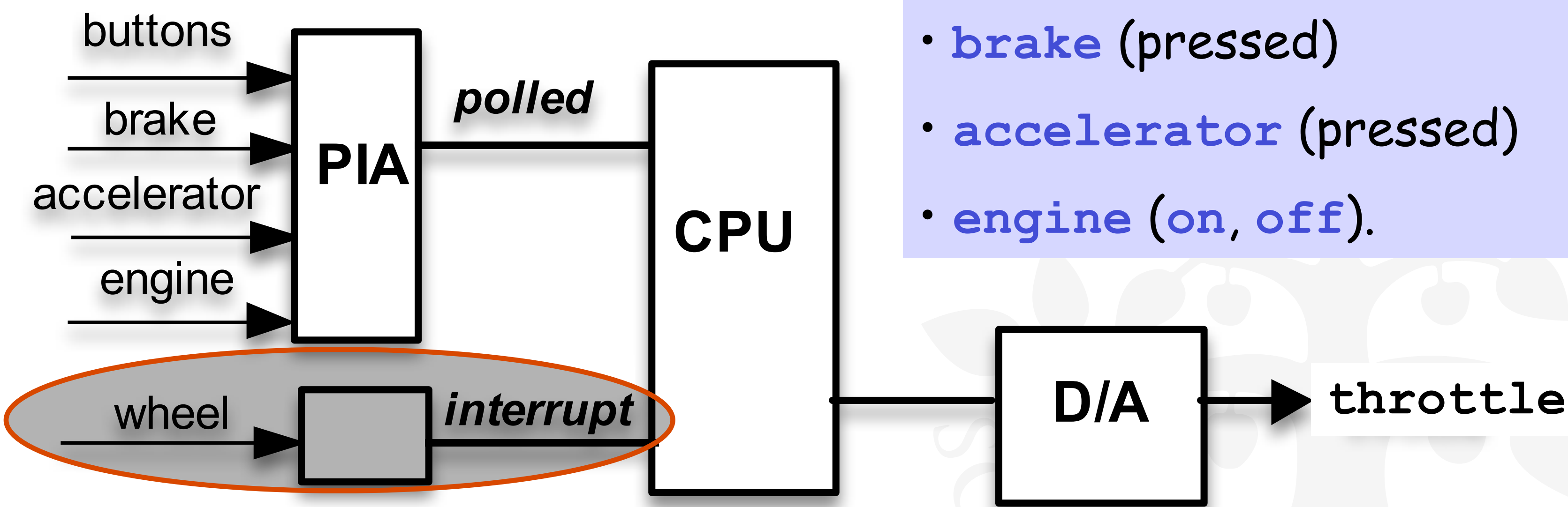
- buttons (`on`, `off`, `resume`)
- `brake` (pressed)
- `accelerator` (pressed)
- `engine` (`on`, `off`).



A 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`).

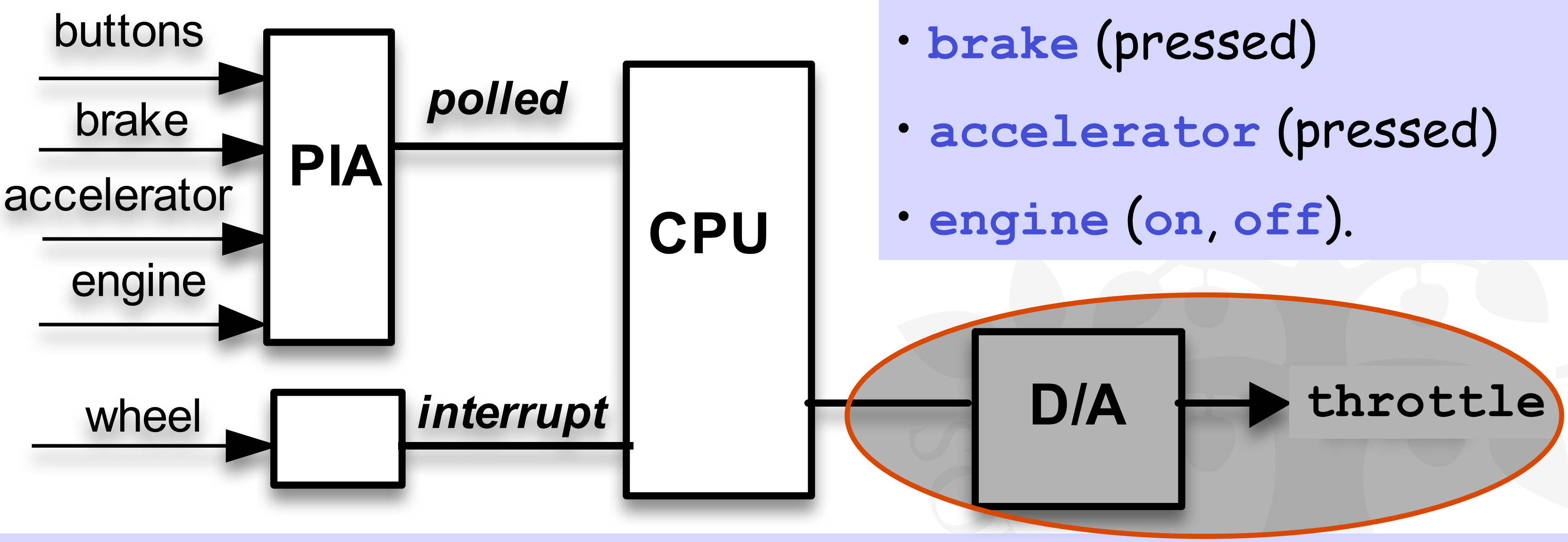


Wheel Revolution Sensor generates interrupts to enable the car `speed` to be calculated.

A Cruise Control System - Hardware

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

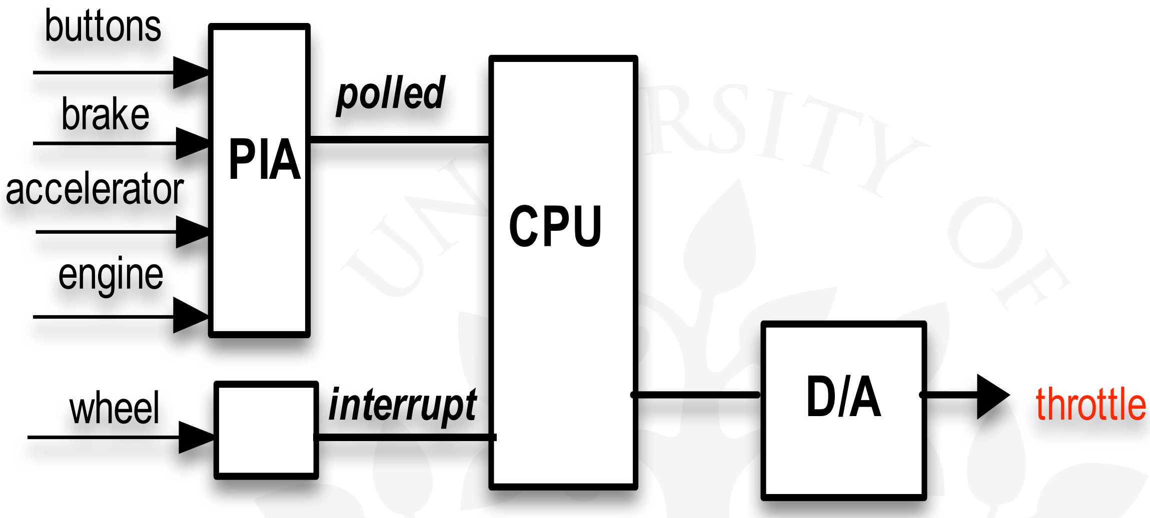
- buttons (on, off, resume)
- brake (pressed)
- accelerator (pressed)
- engine (on, off).



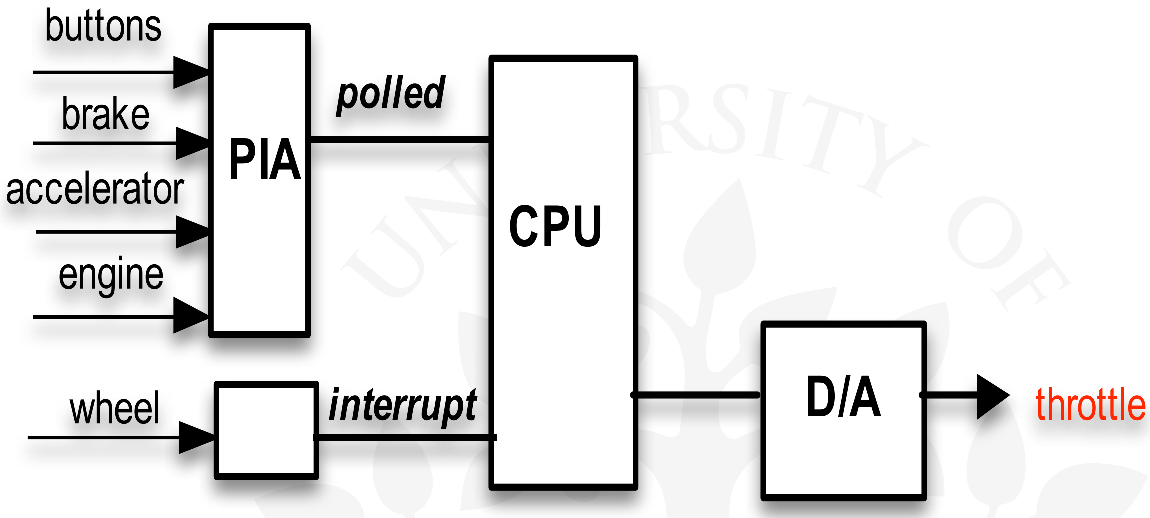
Wheel Revolution Sensor generates interrupts to enable the car speed to be calculated.

Output: The cruise control system controls the car speed by setting the throttle via the digital-to-analogue (D/A) converter.

Model - Design

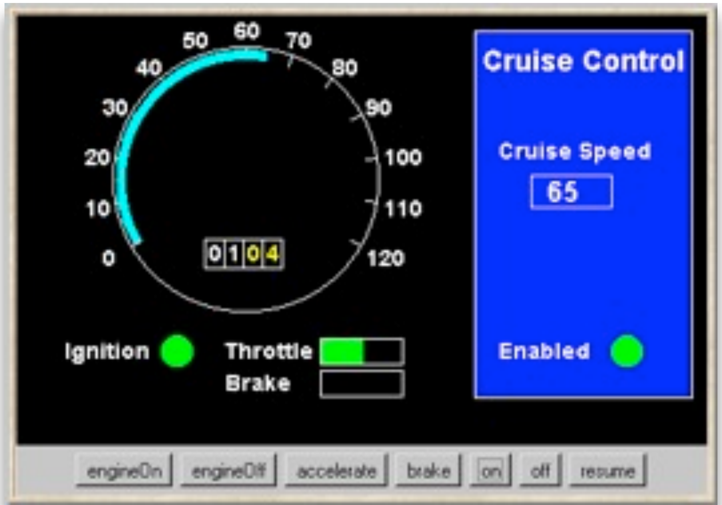


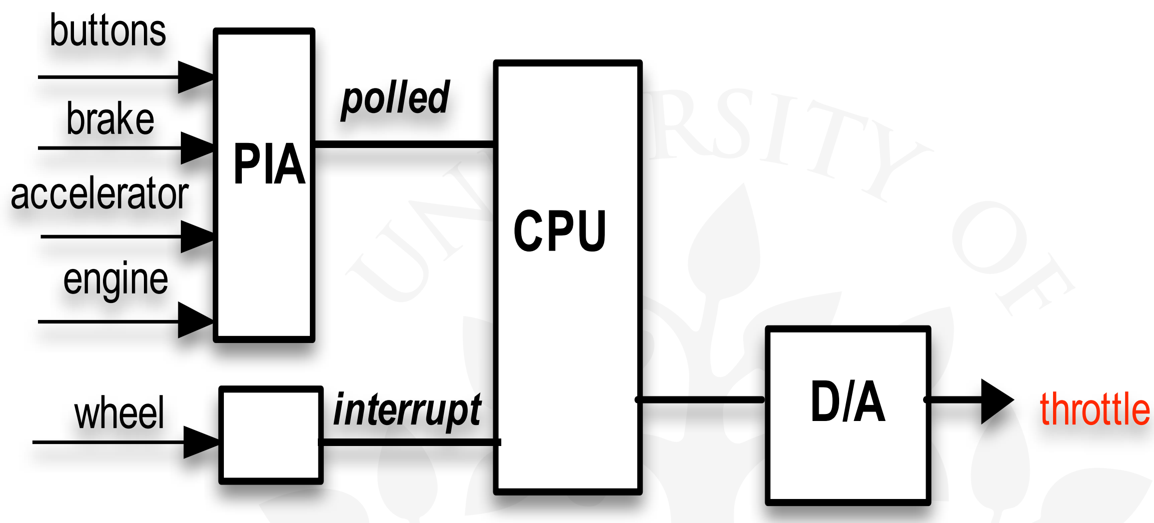
Model - Design



1. Identify the main **events**, **actions**, and **interactions**:

Model - Design



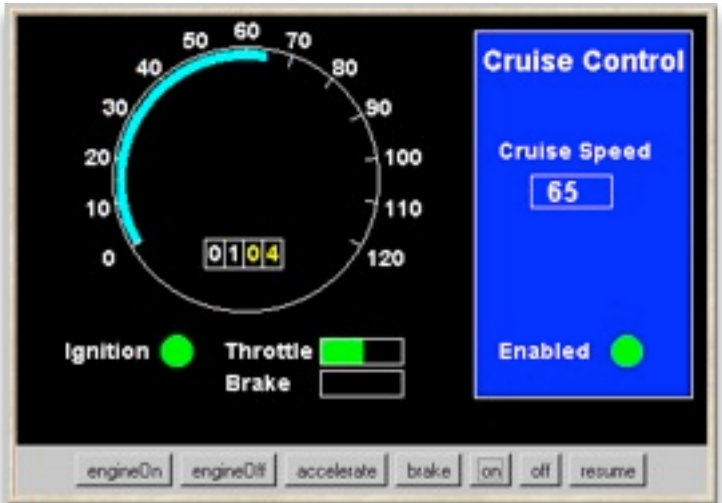


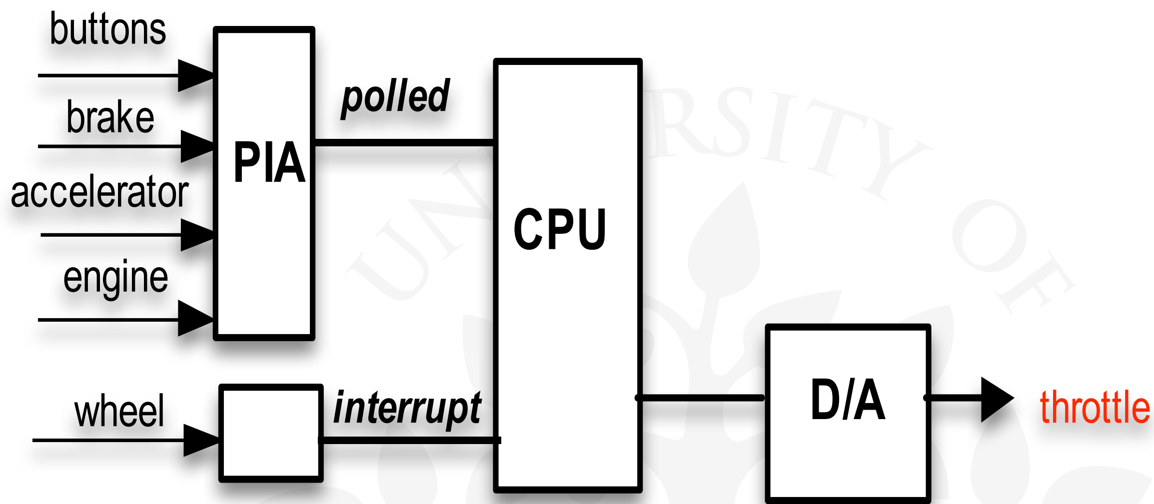
1. Identify the main **events**, **actions**, and **interactions**:

   `on, off, resume,`

   `brake, accelerator, engineOn, engineOff,`

Model - Design





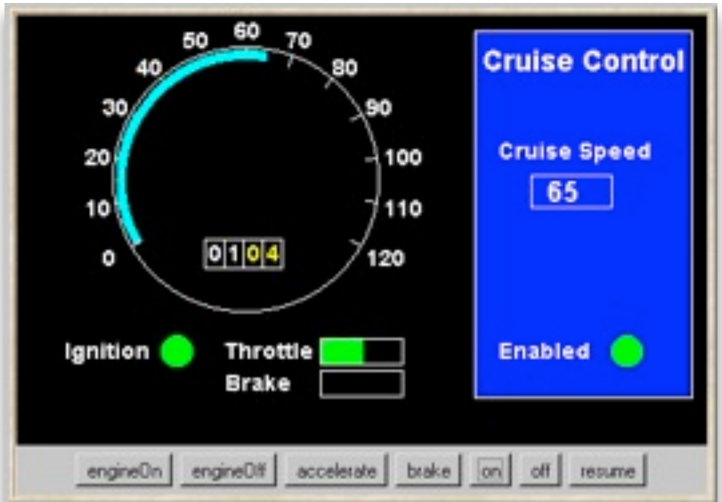
1. Identify the main **events**, **actions**, and **interactions**:

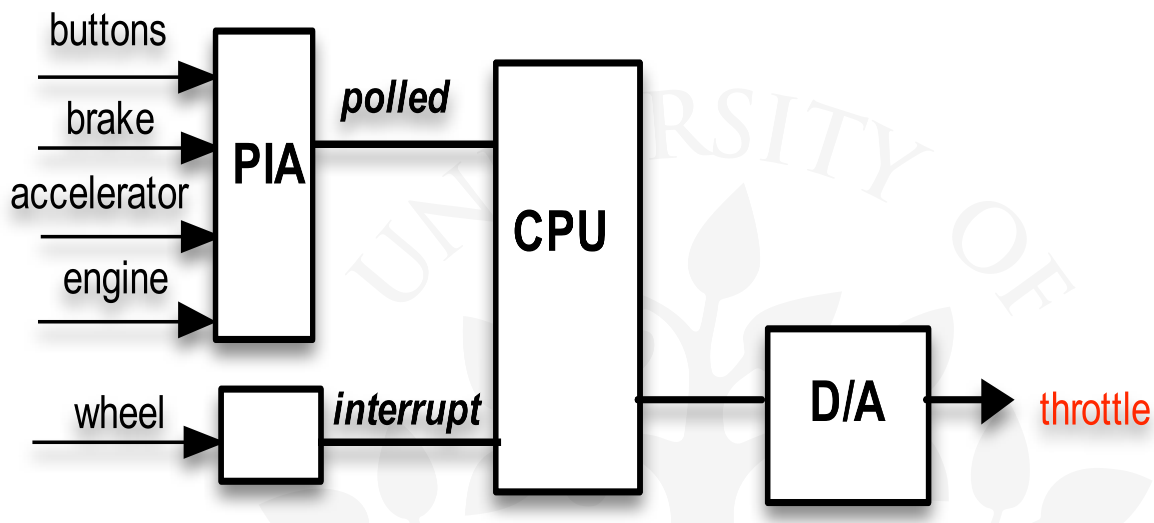
   `on, off, resume,`

   `brake, accelerator, engineOn, engineOff,`

   `speed,`

Model - Design





1. Identify the main **events**, **actions**, and **interactions**:

   on, off, resume,

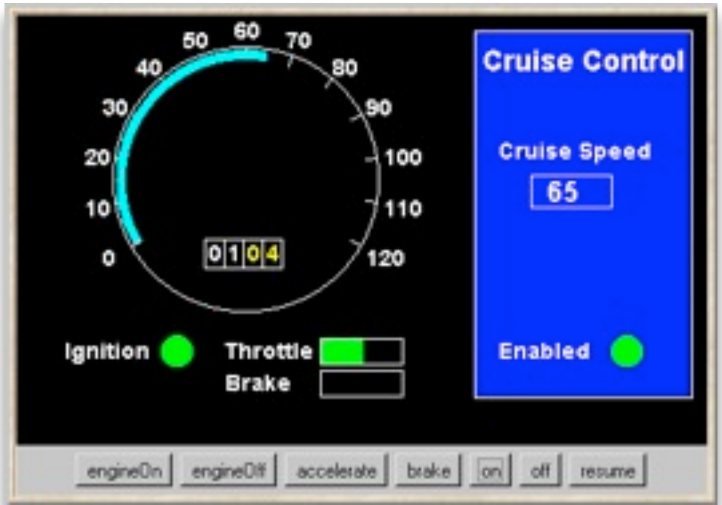
   brake, accelerator, engineOn, engineOff,

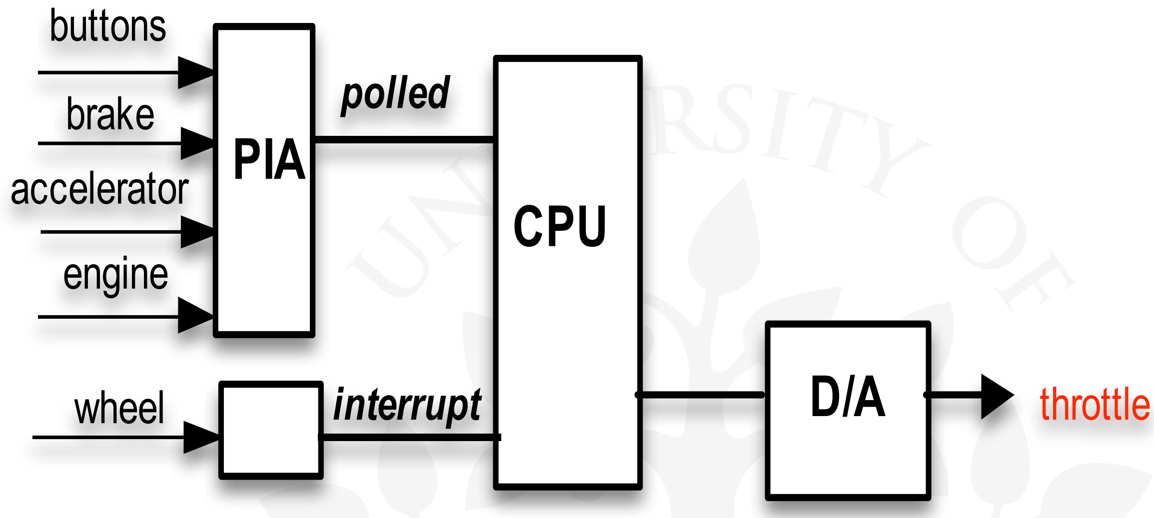
   speed,

   setThrottle, zoom,

Model - Design

University of Southern Denmark





1. Identify the main **events**, **actions**, and **interactions**:

   `on, off, resume,`

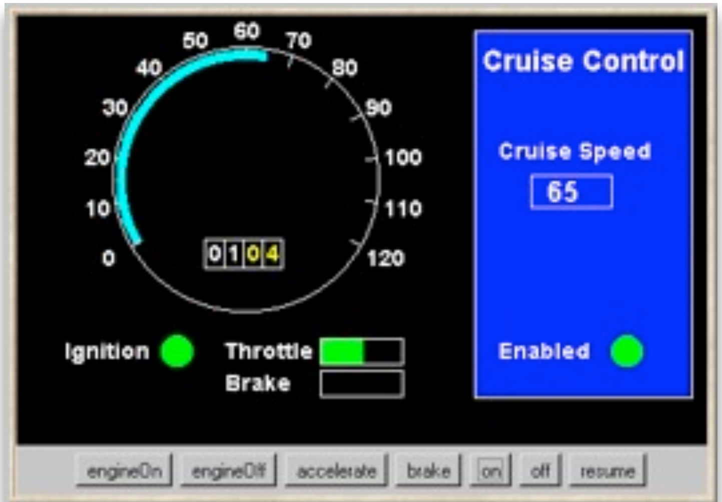
   `brake, accelerator, engineOn, engineOff,`

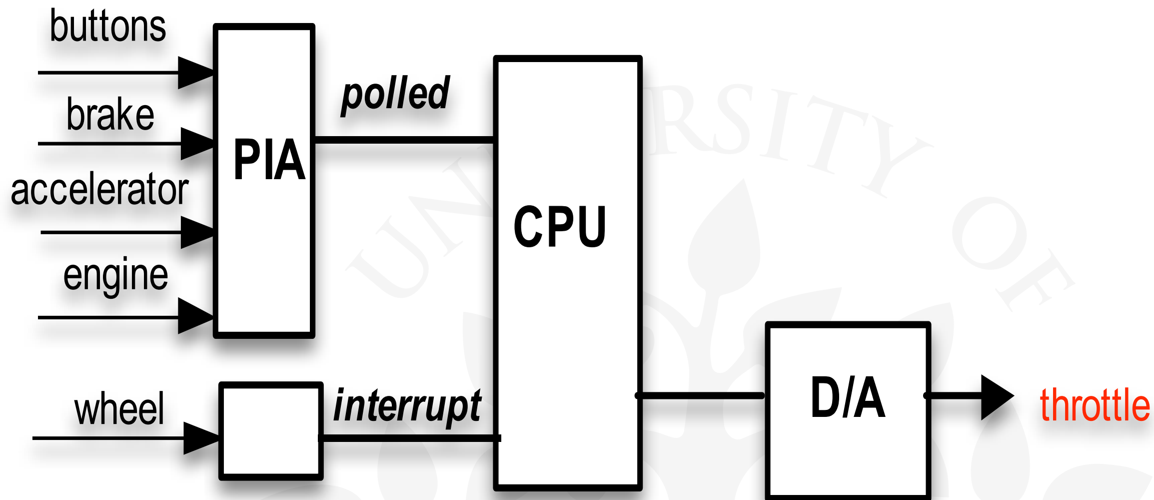
   `speed,`

   `setThrottle, zoom,`

   `clearSpeed, recordSpeed,`

Model - Design





1. Identify the main **events**, **actions**, and **interactions**:
   `on, off, resume,`
   `brake, accelerator, engineOn, engineOff,`
   `speed,`
   `setThrottle, zoom,`
   `clearSpeed, recordSpeed,`
   `enableControl, disableControl.`

Model - Design

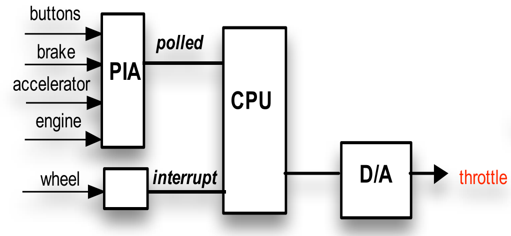
University of Southern Denmark

1. Identify the main **events**, **actions**, and **interactions**:

   `on, off, resume,`
   `brake, accelerator, engineOn, engineOff,`
   `speed,`
   `setThrottle, zoom,`
   `clearSpeed, recordSpeed,`
   `enableControl, disableControl.`

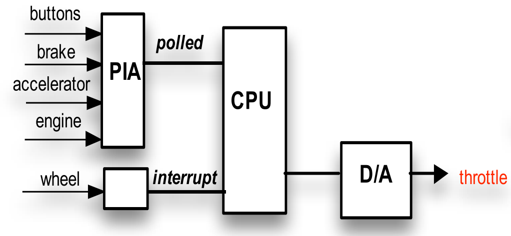
```
set Sensors = {engineOn, engineOff, accelerator, brake,
               on, off, resume}

set Engine  = {engineOn, engineOff}

set Prompts = {enableControl, disableControl,
               clearSpeed, recordSpeed}
```

Model - Outline Design



2. Identify and define the main **processes**:

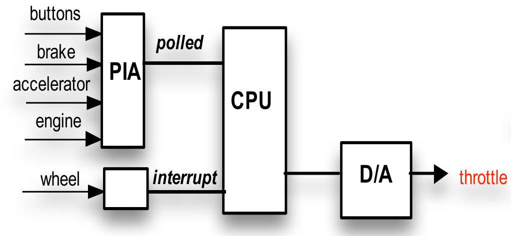
Model - Outline Design



2. Identify and define the main **processes**:

> **Sensor Scan** monitors the buttons, brake, accelerator and engine events

Model - Outline Design



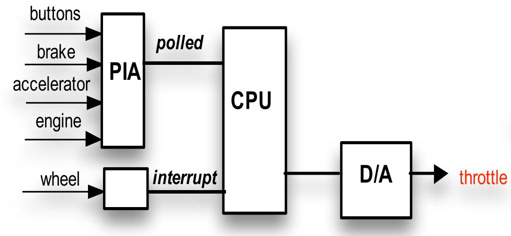
UNIVERSITY OF SOUTHERN DENMARK

2. Identify and define the main **processes**:

Sensor Scan monitors the buttons, brake, accelerator and engine events

Input Speed monitors the speed (when the engine is on), and provides the current speed readings to the speed control

Model - Outline Design



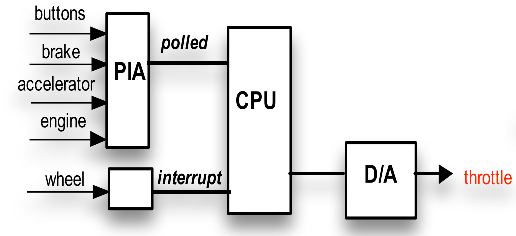
2. Identify and define the main **processes**:

Sensor Scan monitors the buttons, brake, accelerator and engine events

Input Speed monitors the speed (when the engine is on), and provides the current speed readings to the speed control

Throttle sets the actual throttle

Model - Outline Design



2. Identify and define the main **processes**:

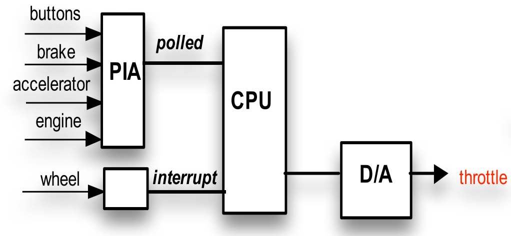
Sensor Scan monitors the buttons, brake, accelerator and engine events

Cruise State Controller triggers **`clearSpeed`** and **`recordSpeed`**, and enables-/disables the speed control

Input Speed monitors the speed (when the engine is on), and provides the current speed readings to the speed control

Throttle sets the actual throttle

Model - Outline Design



2. Identify and define the main **processes**:

Sensor Scan monitors the buttons, brake, accelerator and engine events

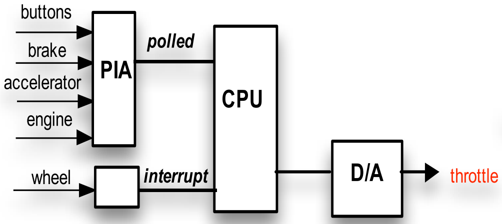
Cruise State Controller triggers `clearSpeed` and `recordSpeed`, and enables-/disables the speed control

Input Speed monitors the speed (when the engine is on), and provides the current speed readings to the speed control

Speed Control clears and records the speed, and sets the throttle accordingly when enabled

Throttle sets the actual throttle

Model - Outline Design



2. Identify and define the main **processes**:

Sensor Scan monitors the buttons, brake, accelerator and engine events

Cruise State Controller triggers **clearSpeed** and **recordSpeed**, and enables-/disables the speed control

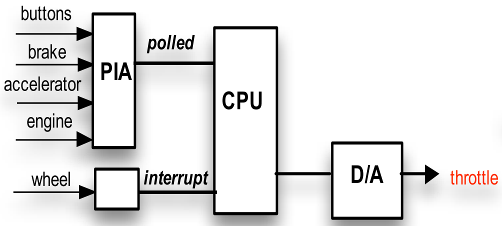
Controlling the state (of the controller)

Input Speed monitors the speed (when the engine is on), and provides the current speed readings to the speed control

Speed Control clears and records the speed, and sets the throttle accordingly when enabled

Throttle sets the actual throttle

Model - Outline Design



2. Identify and define the main **processes**:

Sensor Scan monitors the buttons, brake, accelerator and engine events

Cruise State Controller triggers `clearSpeed` and `recordSpeed`, and enables-/disables the speed control

Controlling the state (of the controller)

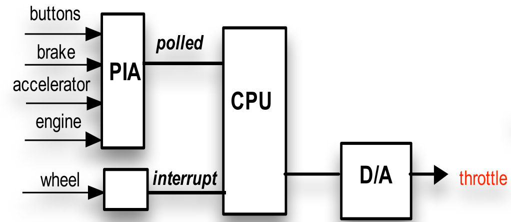
Input Speed monitors the speed (when the engine is on), and provides the current speed readings to the speed control

Speed Control clears and records the speed, and sets the throttle accordingly when enabled

Controlling the throttle

Throttle sets the actual throttle

Model - Outline Design



2. Identify and define the main **processes**:

Sensor Scan monitors the buttons, brake, accelerator and engine events

Cruise State Controller triggers **clearSpeed** and **recordSpeed**, and enables-/disables the speed control

Controlling the state (of the controller)

Input Speed monitors the speed (when the engine is on), and provides the current speed readings to the speed control

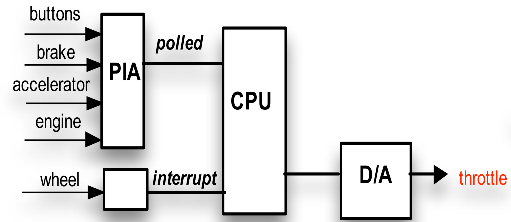
Speed Control clears and records the speed, and sets the throttle accordingly when enabled

Controlling the throttle

Throttle sets the actual throttle

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

Model - Outline Design



UNIVERSITY OF SOUTHERN DENMARK

2. Identify and define the main **processes**:

Sensor Scan monitors the buttons, brake, accelerator and engine events

Engine

Input Speed monitors the speed (when the engine is on), and provides the current speed readings to the speed control

Cruise State Controller triggers **`clearSpeed`** and **`recordSpeed`**, and enables-/disables the speed control

Controlling the state (of the controller)

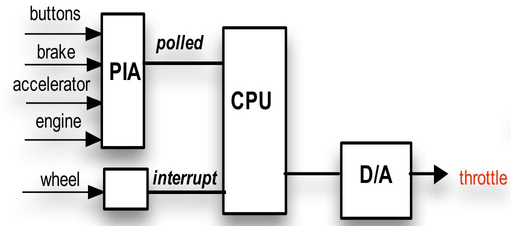
Speed Control clears and records the speed, and sets the throttle accordingly when enabled

Controlling the throttle

Throttle sets the actual throttle

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

Model - Outline Design



UNIVERSITY OF SOUTHERN DENMARK

2. Identify and define the main **processes**:

Sensor Scan monitors the buttons, brake, accelerator and engine events

Sensors

Cruise State Controller triggers `clearSpeed` and `recordSpeed`, and enables-/disables the speed control

Controlling the state (of the controller)

Engine

Input Speed monitors the speed (when the engine is on), and provides the current speed readings to the speed control

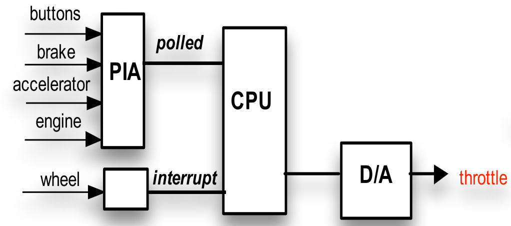
Speed Control clears and records the speed, and sets the throttle accordingly when enabled

Controlling the throttle

Throttle sets the actual throttle

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

Model - Outline Design



2. Identify and define the main **processes**:

Sensor Scan monitors the buttons, brake, accelerator and engine events

→ **Sensors** →

Cruise State Controller triggers **clearSpeed** and **recordSpeed**, and enables-/ disables the speed control

Controlling the state (of the controller)

↓ **Engine**

Input Speed monitors the speed (when the engine is on), and provides the current speed readings to the speed control

→ **speed** →

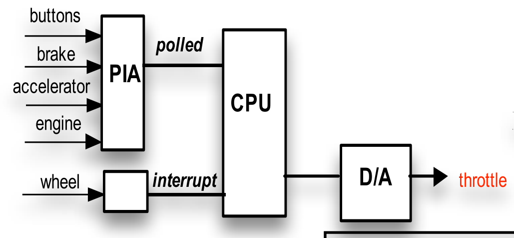
Speed Control clears and records the speed, and sets the throttle accordingly when enabled

Controlling the throttle

Throttle sets the actual throttle

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

Model - Outline Design



University of Southern Denmark

2. Identify and define the main **processes**:

Sensor Scan monitors the buttons, brake, accelerator and engine events

→ **Sensors** →

Cruise State Controller triggers **clearSpeed** and **recordSpeed**, and enables-/disables the speed control

Controlling the state (of the controller)

↓ **Engine**

↓ **Prompts**

Input Speed monitors the speed (when the engine is on), and provides the current speed readings to the speed control

→ **speed** →

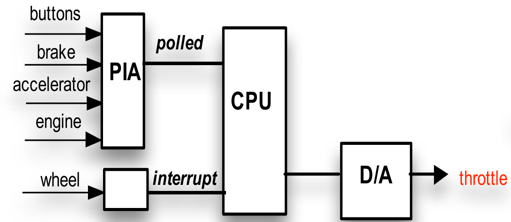
Speed Control clears and records the speed, and sets the throttle accordingly when enabled

Controlling the throttle

Throttle sets the actual throttle

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

Model - Outline Design



2. Identify and define the main **processes**:

Sensor Scan monitors the buttons, brake, accelerator and engine events

Cruise State Controller triggers `clearSpeed` and `recordSpeed`, and enables-/disables the speed control

Controlling the state (of the controller)

Sensors →

↓ Engine

↓ Prompts

Input Speed monitors the speed (when the engine is on), and provides the current speed readings to the speed control

speed →

Speed Control clears and records the speed, and sets the throttle accordingly when enabled

setThrottle →

Throttle sets the actual throttle

Controlling the throttle

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

Model – Design (Step 1+2+3)

Model – Design (Step 1+2+3)

1. Identify the main **events**, **actions**, and **interactions**:

`on`, `off`, `resume`, `brake`, `accelerator`

`engineOn`, `engineOff`,

} **Sensors**

`speed`, `setThrottle`

`clearSpeed`, `recordSpeed`,

`enableControl`, `disableControl`

} **Prompts**

Model – Design (Step 1+2+3)

University of Southern Denmark

1. Identify the main **events**, **actions**, and **interactions**:

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

   `speed, setThrottle`
   `clearSpeed,recordSpeed,`
   `enableControl,disableControl` } **Prompts**

2. Identify and define the main **processes**:

   `Sensor Scan, Input Speed,`
   `Cruise Controller, Speed Control, and`
   `Throttle`

Model – Design (Step 1+2+3)

UNIVERSITY OF SOUTHERN DENMARK

1. Identify the main **events**, **actions**, and **interactions**:

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

   `speed, setThrottle`
   `clearSpeed,recordSpeed,`
   `enableControl,disableControl` } **Prompts**

2. Identify and define the main **processes**:

   `Sensor Scan, Input Speed,`
   `Cruise Controller, Speed Control, and`
   `Throttle`

3. Identify and define the main **properties** of interest:

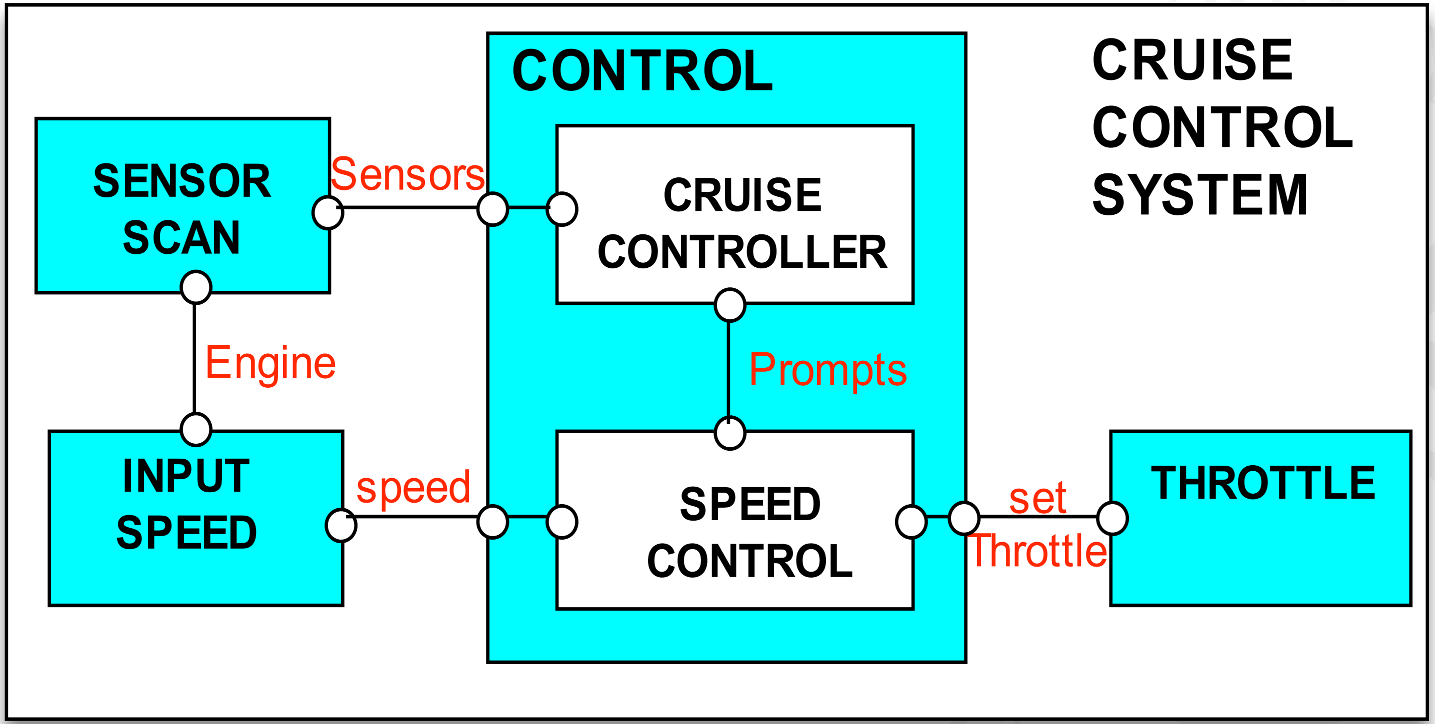
   safety: cruise control disabled when
   `off`, `brake` or `accelerator` is pressed.

Model - Design (Step 4)

4. Structure the processes into an **architecture**:

Model - Design (Step 4)

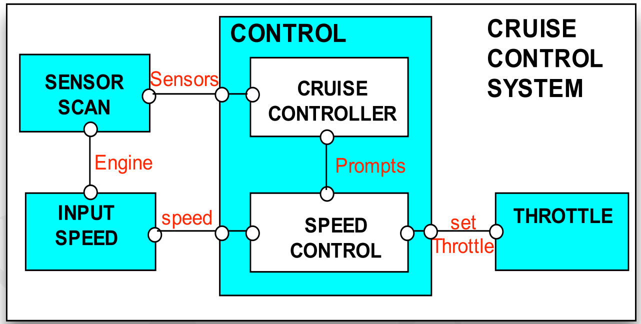
4. Structure the processes into an **architecture**:



The CONTROL system is structured as two processes:

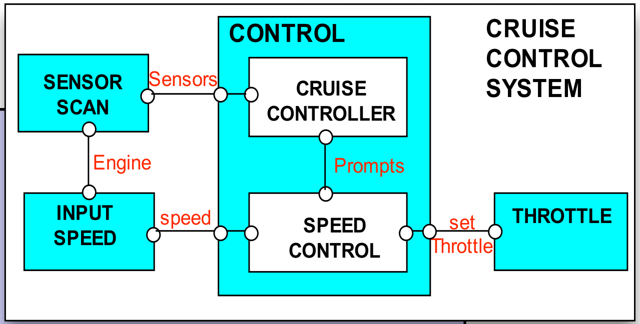
- **`CRUISECONTROLLER`** (controlling the state); and
- **`SPEEDCONTROL`** (controlling the throttle)

Model Elaboration - Process Definitions




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

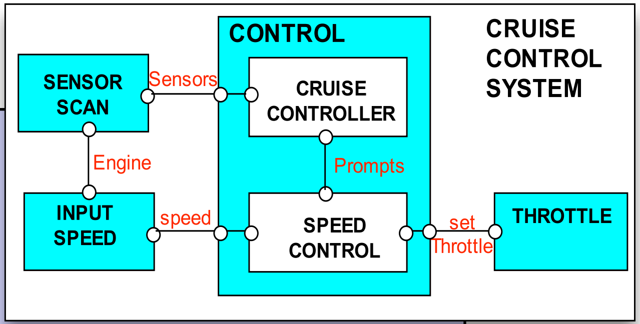
Model Elaboration - Process Definitions




```
SENSORSCAN = ({Sensors} -> SENSORSCAN).
```

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

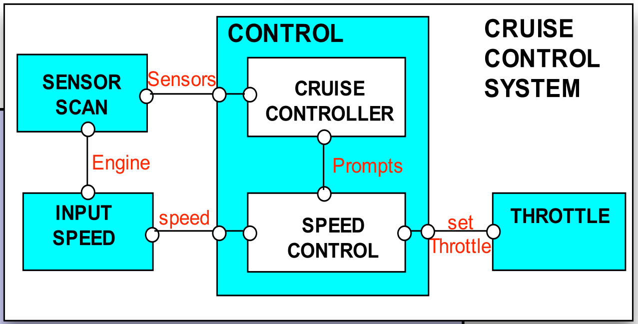
Model Elaboration - Process Definitions




```
SENSORSCAN = ({Sensors} -> SENSORSCAN).

      // monitor speed when engine on
INPUTSPEED = (engineOn  -> CHECKSPEED),
CHECKSPEED = (speed     -> CHECKSPEED  // monitor speed
             |engineOff -> INPUTSPEED).
```

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

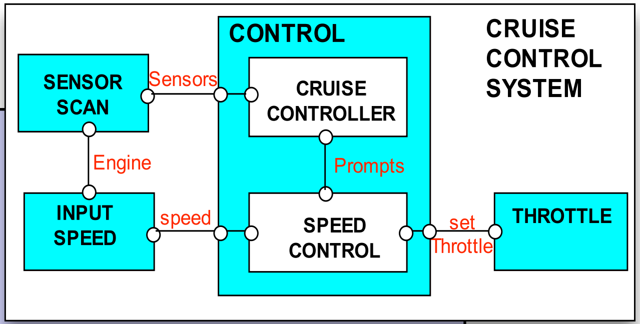
Model Elaboration - Process Definitions




```
SENSORSCAN = ({Sensors} -> SENSORSCAN).

        // monitor speed when engine on
INPUTSPEED = (engineOn  -> CHECKSPEED),
CHECKSPEED = (speed     -> CHECKSPEED  // monitor speed
             |engineOff -> INPUTSPEED).

        // zoom when throttle set
THROTTLE =(setThrottle -> zoom -> THROTTLE).
```

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

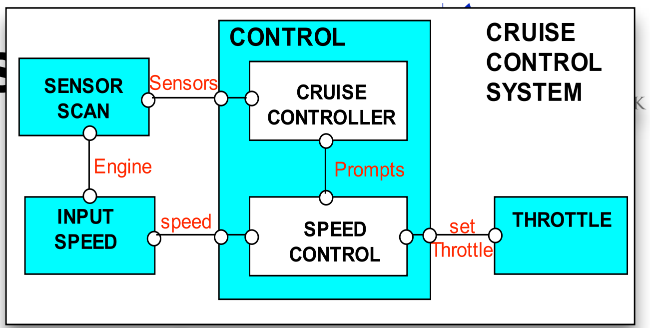
Model Elaboration - Process Definitions



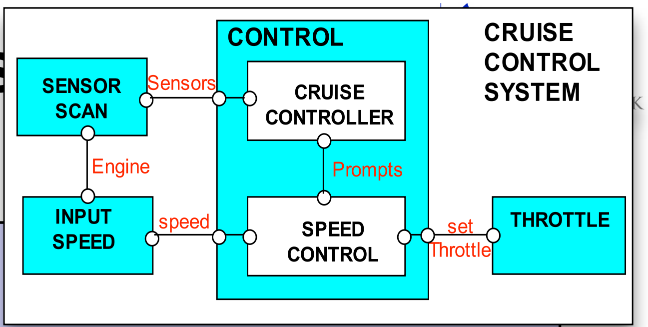

```
SENSORSCAN = ({Sensors} -> SENSORSCAN).

      // monitor speed when engine on
INPUTSPEED = (engineOn  -> CHECKSPEED),
CHECKSPEED = (speed     -> CHECKSPEED  // monitor speed
             |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 Sensors = {engineOn, engineOff, accelerator, brake, on, off, resume}
set Engine  = {engineOn, engineOff}
set Prompts = {enableControl, disableControl, clearSpeed, recordSpeed}
```

Model Elaboration - Process Definitions

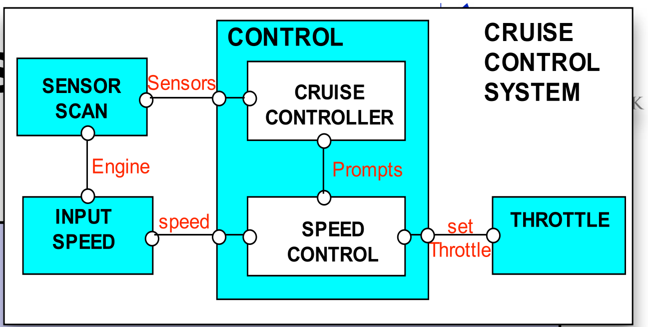


Model Elaboration - Process Definitions



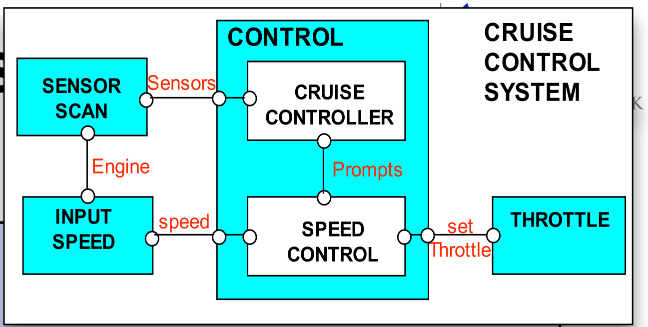

```
set DisableActions={off,brake,accelerator}
CRUISECONTROLLER = INACTIVE,
```

Model Elaboration - Process Definitions



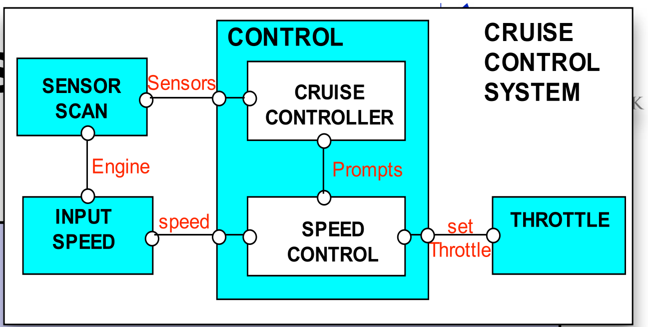

```
set DisableActions={off,brake,accelerator}
CRUISECONTROLLER = INACTIVE,

INACTIVE =(engineOn -> clearSpeed           -> ACTIVE
          | DisableActions ->               -> INACTIVE),
```

Model Elaboration - Process Definitions



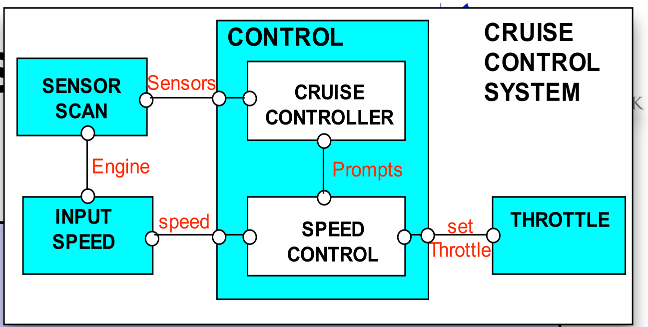

```
set DisableActions={off,brake,accelerator}
CRUISECONTROLLER = INACTIVE,

INACTIVE =(engineOn -> clearSpeed              -> ACTIVE
          | DisableActions ->                  -> INACTIVE),
ACTIVE   =(engineOff                           -> INACTIVE
          |on-> recordSpeed-> enableControl-> CRUISING
          |DisableActions                     -> ACTIVE),
```

Model Elaboration - Process Definitions



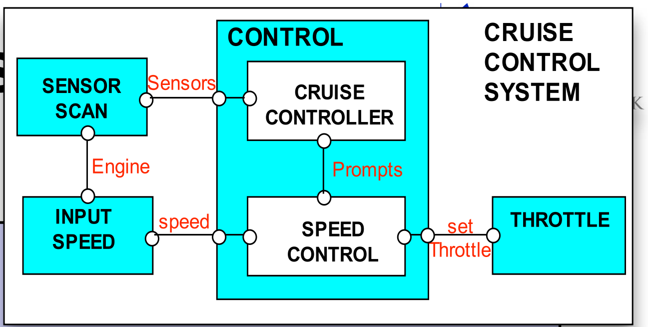

```
set DisableActions={off,brake,accelerator}
CRUISECONTROLLER = INACTIVE,

INACTIVE =(engineOn -> clearSpeed            -> ACTIVE
          | DisableActions ->                -> INACTIVE),
ACTIVE   =(engineOff                         -> INACTIVE
          |on-> recordSpeed-> enableControl-> CRUISING
          |DisableActions                    -> ACTIVE),
    // enable speed control when cruising,
    // disable when off, brake, or accelerator pressed
```

Model Elaboration - Process Definitions



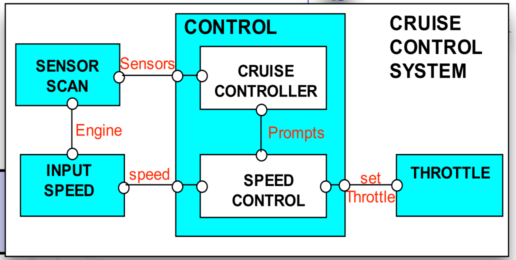

```
set DisableActions={off,brake,accelerator}
CRUISECONTROLLER = INACTIVE,

INACTIVE =(engineOn -> clearSpeed                -> ACTIVE
          | DisableActions ->                    -> INACTIVE),
ACTIVE   =(engineOff                             -> INACTIVE
          |on-> recordSpeed-> enableControl-> CRUISING
          |DisableActions                        -> ACTIVE),
     // enable speed control when cruising,
     // disable when off, brake, or accelerator pressed
CRUISING =(engineOff                             -> INACTIVE
          |DisableActions  ->disableControl-> STANDBY
          |on->recordSpeed -> enableControl-> CRUISING),
```

Model Elaboration - Process Definitions



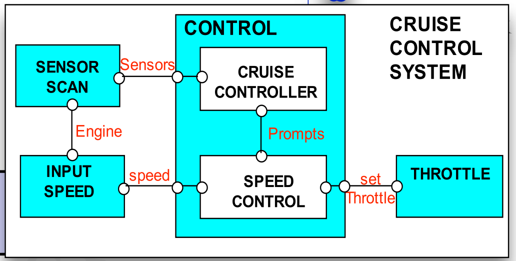

```
set DisableActions={off,brake,accelerator}
CRUISECONTROLLER = INACTIVE,

INACTIVE =(engineOn -> clearSpeed             -> ACTIVE
          | DisableActions ->                 -> INACTIVE),
ACTIVE   =(engineOff                          -> INACTIVE
          |on-> recordSpeed-> enableControl-> CRUISING
          |DisableActions                     -> ACTIVE),
     // enable speed control when cruising,
     // disable when off, brake, or accelerator pressed
CRUISING =(engineOff                          -> INACTIVE
          |DisableActions  ->disableControl-> STANDBY
          |on->recordSpeed -> enableControl-> CRUISING),
STANDBY  =(engineOff                          -> INACTIVE
          |resume          -> enableControl-> CRUISING
          |on-> recordSpeed-> enableControl-> CRUISING
          |DisableActions  ->                  STANDBY).
```

Model - CONTROL Sub-System




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

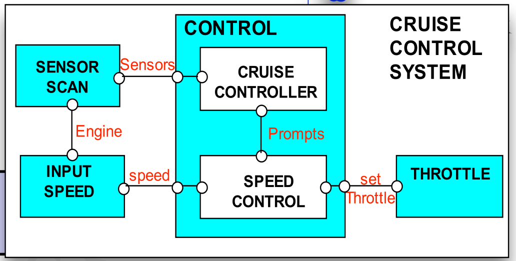
Model - CONTROL Sub-System




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

Testing (animation in LTSA) will check particular traces

Model - CONTROL Sub-System

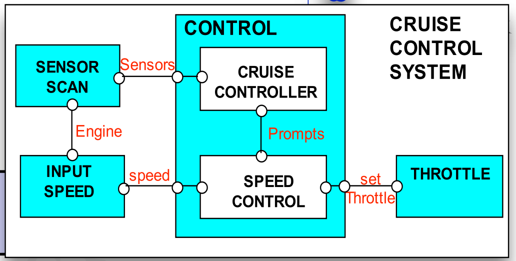



```
||CONTROL = (CRUISECONTROLLER||SPEEDCONTROL).
```

Testing (animation in LTSA) will check particular traces

- Is control enabled after the engine is switched on and the `on` button is pressed?

Model - CONTROL Sub-System

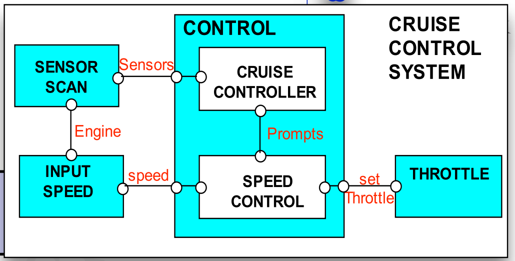



```
||CONTROL = (CRUISECONTROLLER||SPEEDCONTROL).
```

Testing (animation in LTSA) will check particular 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?

Model - CONTROL Sub-System

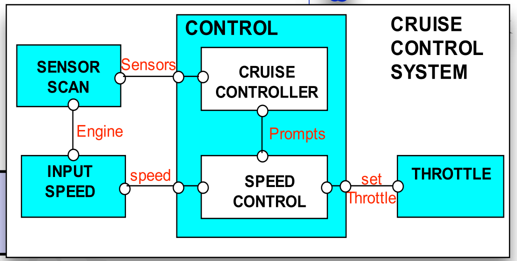



```
||CONTROL = (CRUISECONTROLLER||SPEEDCONTROL).
```

Testing (animation in LTSA) will check particular 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** is then pressed?

Model - CONTROL Sub-System



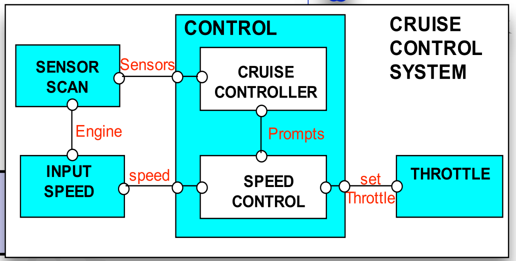

```
||CONTROL = (CRUISECONTROLLER||SPEEDCONTROL).
```

Testing (animation in LTSA) will check particular 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` is then pressed?

Verification (aka. model-checking) will exhaustively analyze all possible traces.

Model - CONTROL Sub-System




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

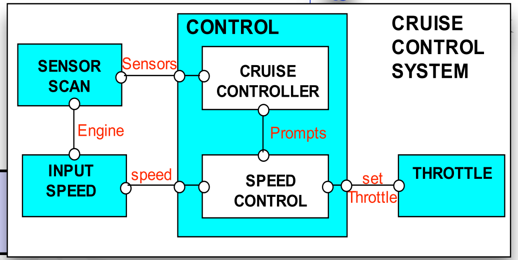
Testing (animation in LTSA) will check particular traces

- 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` is then pressed?

Verification (aka. model-checking) will exhaustively analyze all possible traces.

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

Model - CONTROL Sub-System




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

Testing (animation in LTSA) will check particular 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` is then pressed?

Verification (aka. model-checking) will exhaustively analyze all possible traces.

Safety:	Is the control always disabled when `off`, `brake`, or `accelerator` is pressed?
Progress:	Can every action eventually be selected?

Model - Safety Properties

Safety checks are compositional.

```
/* safety */ property S = ...
```

If there is no violation at a sub-system level, then there cannot be a violation when the sub-system is composed with other sub-systems.

```
P |= S   ∧   Q |= S      =>      (P||Q)  |= S
```

This is because, if the **ERROR** state of a particular safety property is unreachable in the LTS of the sub-system, it remains unreachable in any subsequent parallel composition which includes the sub-system.

Model - Safety Properties


```
/* safety */ property S = ...
```

Safety checks are compositional.

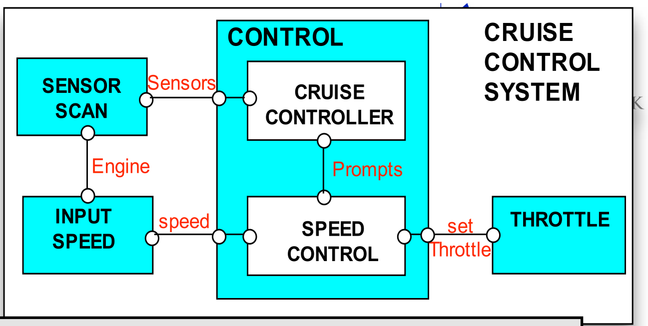
If there is no violation at a sub-system level, then there cannot be a violation when the sub-system is composed with other sub-systems.

```
P |= S   ∧   Q |= S      =>      (P||Q) |= S
```

This is because, if the **ERROR** state of a particular safety property is unreachable in the LTS of the sub-system, it remains unreachable in any subsequent parallel composition which includes the sub-system.

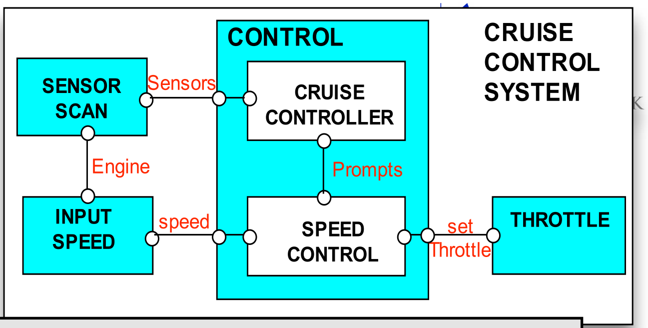
Thus: Safety properties should be composed with the appropriate (sub-)system to which the property refers.

Model - Safety Properties



Is the control always disabled when `off`/`brake`/`acc` pressed?

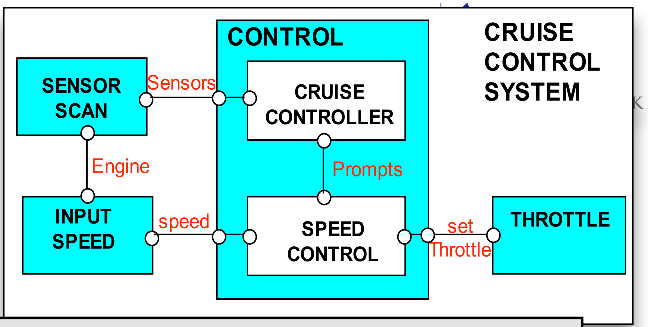
Model - Safety Properties



Is the control always disabled when `off`/`brake`/`acc` pressed?

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

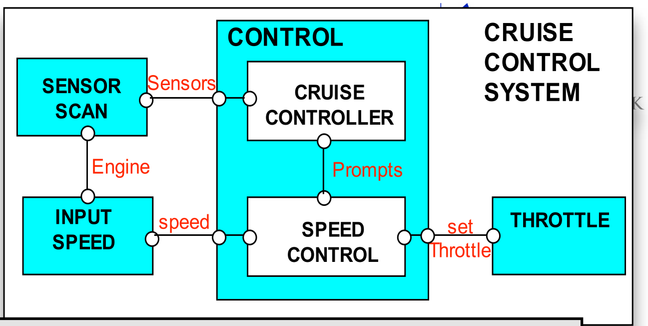
Model - Safety Properties



Is the control always disabled when off/brake/acc pressed?

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

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

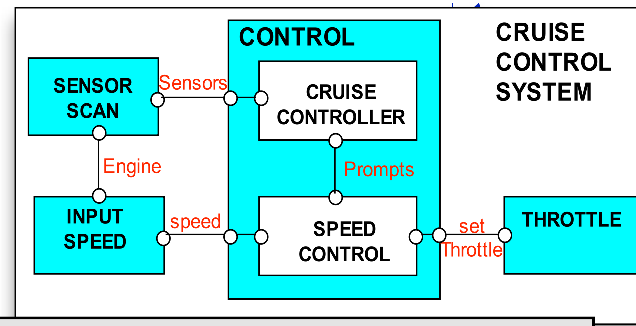
Model - Safety Properties



Is the control always disabled when off/brake/acc pressed?

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

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


SAFETYACTION = (disableControl->CRUISESAFETY).
```

Model - Safety Properties



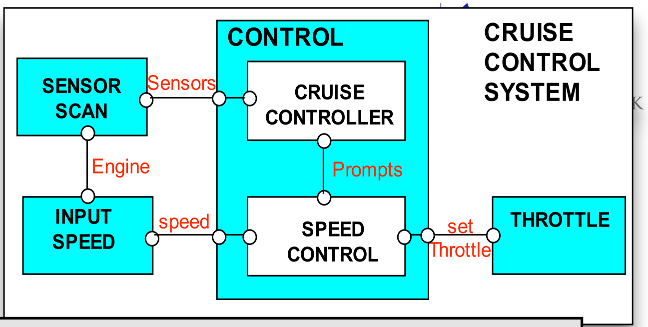
Is the control always disabled when off/brake/acc pressed?

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

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


SAFETYACTION = (disableControl->CRUISESAFETY).
```

Composition with CONTROL processes:

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

Model - Safety Properties



Is the control always disabled when `off`/`brake`/`acc` pressed?

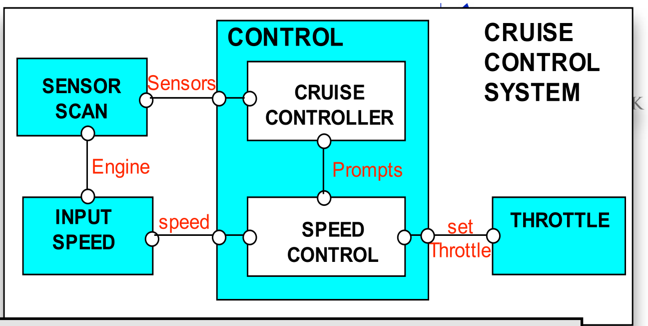
```
property CRUISESAFETY =
      ( {on,resume} -> SAFETYCHECK,
      | {DisableActions,disableControl} -> CRUISESAFETY),

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


SAFETYACTION = (disableControl->CRUISESAFETY).
```

Composition with CONTROL processes:

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

Verify **CRUISESAFETY**?

Model - Safety Properties



Is the control always disabled when off/brake/acc pressed?

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

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


SAFETYACTION = (disableControl->CRUISESAFETY).
```

Composition with CONTROL processes:

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

Verify CRUISESAFETY? ☹

Model - Safety Properties

Safety analysis using LTSA produces the following **violation**:

```
Trace to property violation in CRUISESAFETY:
     engineOn
     clearSpeed
     on
     recordSpeed
     enableControl
     engineOff
     off
     off
```

Model - Safety Properties

Safety analysis using LTSA produces the following **violation**:

```
Trace to property violation in CRUISESAFETY:
     engineOn
     clearSpeed
     on
     recordSpeed
     enableControl
     engineOff
     off
     off
```

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.

Model - Progress Properties

Progress checks are **not** compositional !

Model - Progress Properties

Progress checks are **not** compositional !

Even if there is no progress violation at a sub-system level, a progress violation may "appear" when the sub-system is composed with other sub-systems.

Model - Progress Properties

Progress checks are **not compositional** !

Even if there is no progress violation at a sub-system level, a progress violation may "appear" when the sub-system is composed with other sub-systems.

```
P |= L  ∧  Q |= L    =>    (P||Q) |= L
```

Model - Progress Properties

Progress checks are **not** compositional !

Even if there is no progress violation at a sub-system level, a progress violation may "appear" when the sub-system is composed with other sub-systems.

~~`P |= L ∧ Q |= L => (P||Q) |= L`~~

Model - Progress Properties

Progress checks are **not compositional** !

Even if there is no progress violation at a sub-system level, a progress violation may "appear" when the sub-system is composed with other sub-systems.

~~`P |= L ∧ Q |= L => (P||Q) |= L`~~

This is because an action in the sub-system may satisfy progress yet be unreachable when the sub-system is composed with other sub-systems which constrain system behavior.

Model - Progress Properties

Progress checks are **not** compositional !

Even if there is no progress violation at a sub-system level, a progress violation may "appear" when the sub-system is composed with other sub-systems.

~~P |= L ∧ Q |= L => (P||Q) |= L~~

This is because an action in the sub-system may satisfy progress yet be unreachable when the sub-system is composed with other sub-systems which constrain system behavior.

However,

we have that:

Model - Progress Properties

Progress checks are **not compositional** !

Even if there is no progress violation at a sub-system level, a progress violation may "appear" when the sub-system is composed with other sub-systems.

~~`P |= L ∧ Q |= L => (P||Q) |= L`~~

This is because an action in the sub-system may satisfy progress yet be unreachable when the sub-system is composed with other sub-systems which constrain system behavior.

However, we have that:

Assume $L \subseteq \alpha(P)$, $L \subseteq \alpha(Q)$:
`(P||Q) |= L => P |= L ∧ Q |= L`

Model - Progress Properties

Progress checks are **not compositional** !

Even if there is no progress violation at a sub-system level, a progress violation may "appear" when the sub-system is composed with other sub-systems.

~~`P |= L ∧ Q |= L => (P||Q) |= L`~~

This is because an action in the sub-system may satisfy progress yet be unreachable when the sub-system is composed with other sub-systems which constrain system behavior.

However, we have that:

`Assume L ⊆ α(P), L ⊆ α(Q):`
`(P||Q) |= L => P |= L ∧ Q |= L`

Thus: Progress checks should be conducted on the complete target system after satisfactory completion of the safety checks.

Model - Progress Properties

Progress check: ||CONTROL = (CRUISECONTROLLER||SPEEDCONTROL).

Model - Progress Properties

Progress check: ||CONTROL = (CRUISECONTROLLER||SPEEDCONTROL).

```
Progress violation for actions:
{accelerator, brake, clearSpeed, disableControl, enableControl,
engineOff, engineOn, off, on, recordSpeed, resume}
Path to terminal set of states:
      engineOn
      clearSpeed
      on
      recordSpeed
      enableControl
      engineOff
      engineOn
Actions in terminal set:
{speed, setThrottle}
```

Model - Progress Properties

Progress check: ||CONTROL = (CRUISECONTROLLER||SPEEDCONTROL).

```
Progress violation for actions:
{accelerator, brake, clearSpeed, disableControl, enableControl,
engineOff, engineOn, off, on, recordSpeed, resume}
Path to terminal set of states:
      engineOn
      clearSpeed
      on
      recordSpeed
      enableControl
      engineOff
      engineOn
Actions in terminal set:
{speed, setThrottle}
```

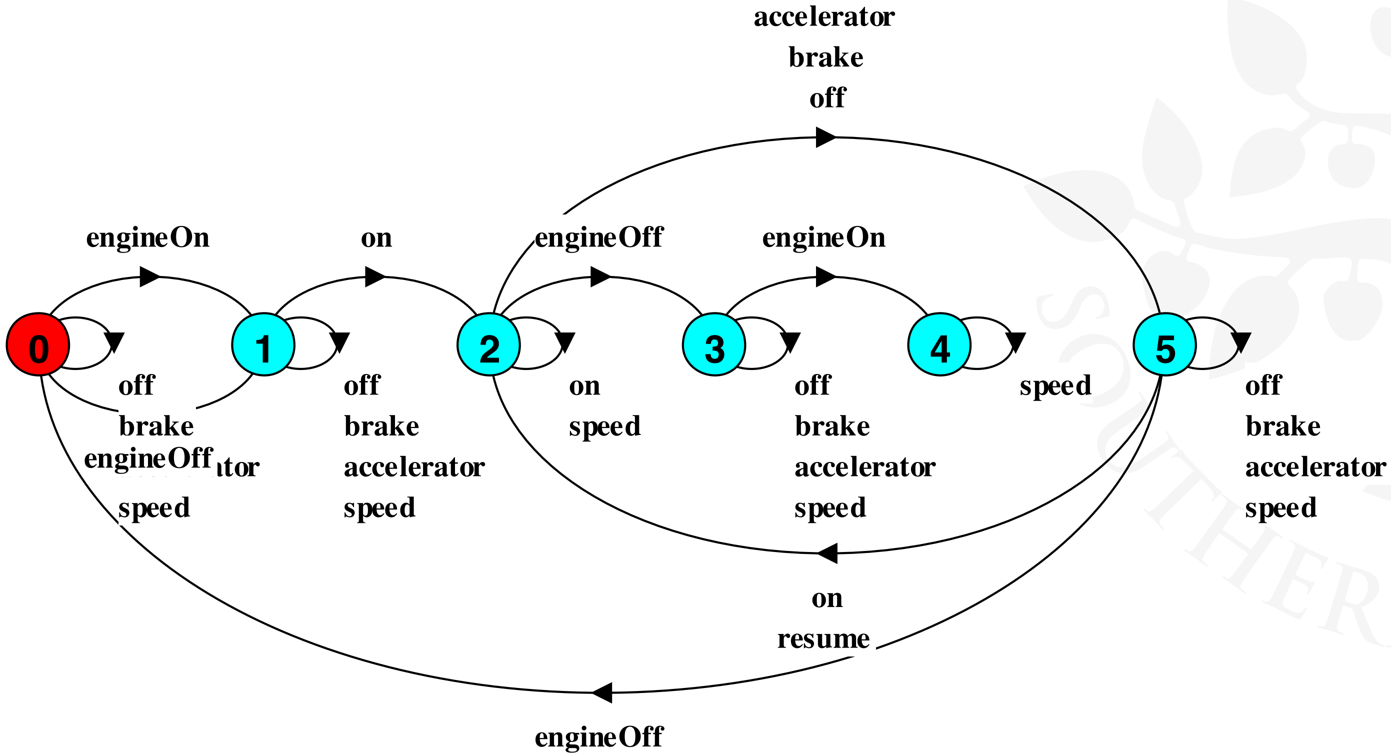
When the engine is switched off:

- CruiseController becomes inactive, whereas
- SpeedControl is not disabled!

Cruise Control Model - Minimized LTS

```
Progress violation trace: engineOn -> clearSpeed -> on ->
recordSpeed -> enableControl -> engineOff -> engineOn.
```

```
||CRUISEMINIMIZED = (CRUISECONTROLSYSTEM)
                    @ {Sensors,speed}.
```



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

Model – Revised Cruise Control System

Fix **CRUISECONTROLLER** so that it disables the **SPEEDCONTROLLER** when the engine is switched off:

```
     // enable speed control when cruising,
     // disable when off, brake, or accelerator pressed
     //          or when the engine is turned off!!!
CRUISING =(engineOff                         -> INACTIVE
          |disableActions-> disableControl -> STANDBY
          |on-> recordSpeed-> enableControl-> CRUISING),
```

Model – Revised Cruise Control System

Fix CRUISECONTROLLER so that it disables the SPEEDCONTROLLER when the engine is switched off:

```
     // enable speed control when cruising,
     // disable when off, brake, or accelerator pressed
     //         or when the engine is turned off!!!
CRUISING =(engineOff   -> disableControl   -> INACTIVE
          |disableActions-> disableControl -> STANDBY
          |on-> recordSpeed-> enableControl-> CRUISING),
```

Model – Revised Cruise Control System

Fix CRUISECONTROLLER so that it disables the SPEEDCONTROLLER when the engine is switched off:

```
     // enable speed control when cruising,
     // disable when off, brake, or accelerator pressed
     //          or when the engine is turned off!!!
CRUISING =(engineOff   -> disableControl   -> INACTIVE
          |disableActions-> disableControl -> STANDBY
          |on-> recordSpeed-> enableControl-> CRUISING),
```

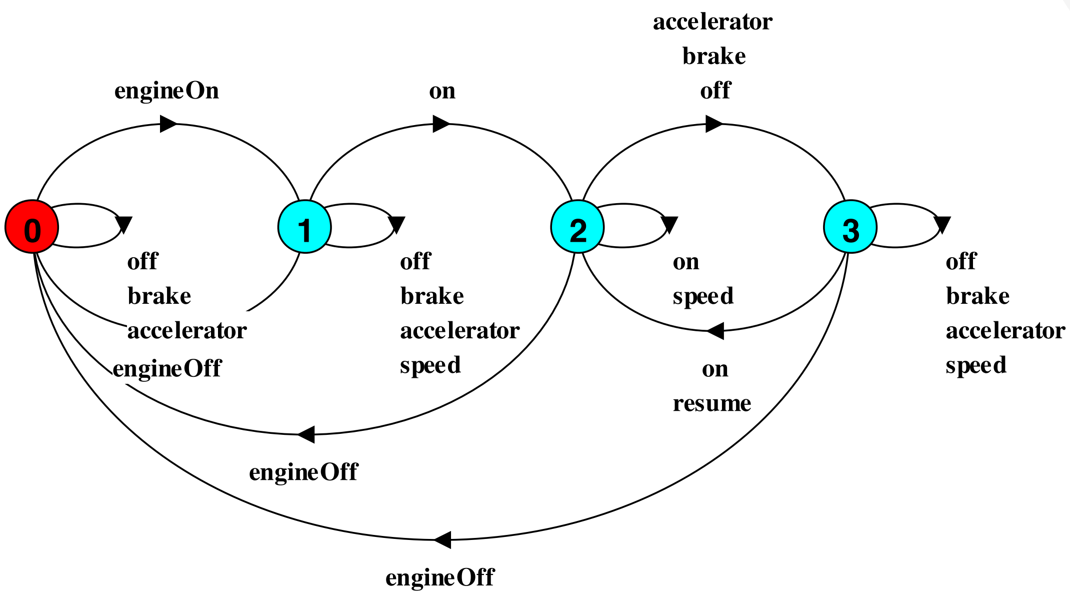
OK now?

Model – Revised Cruise Control System (Properties)

```
property CRUISESAFETYv2 =
    ({off,accelerator,...,engineOff} -> CRUISESAFETYv2
                |{on,resume}           -> SAFETYCHECK),
SAFETYCHECK   = ({on,resume}           -> SAFETYCHECK
                |{off,...,engineOff}   -> SAFETYACTION
                |disableControl        -> CRUISESAFETYv2),
SAFETYACTION = (disableControl         -> CRUISESAFETYv2).
```

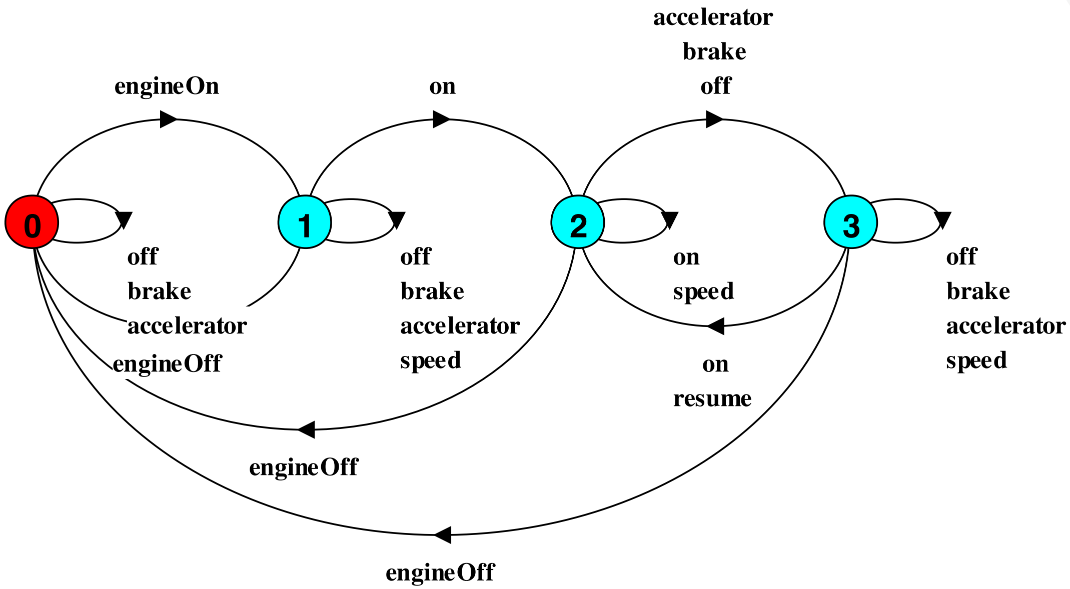
Model – Revised Cruise Control System (Properties)


```
property CRUISESAFETYv2 =
    ({off,accelerator,...,engineOff} -> CRUISESAFETYv2
                |{on,resume}          -> SAFETYCHECK),
SAFETYCHECK   = ({on,resume}          -> SAFETYCHECK
                |{off,...,engineOff}  -> SAFETYACTION
                |disableControl       -> CRUISESAFETYv2),
SAFETYACTION = (disableControl        -> CRUISESAFETYv2).
```



Model – Revised Cruise Control System (Properties)

```
property CRUISESAFETYv2 =
    ({off,accelerator,...,engineOff} -> CRUISESAFETYv2
                |{on,resume}         -> SAFETYCHECK),
SAFETYCHECK   = ({on,resume}         -> SAFETYCHECK
                |{off,...,engineOff} -> SAFETYACTION
                |disableControl      -> CRUISESAFETYv2),
SAFETYACTION = (disableControl       -> CRUISESAFETYv2).
```




```
No deadlocks/errors
```

```
No progress violations
detected
```

Model - System Sensitivities (under *Adverse Conditions*)

```
||SPEEDHIGH = CRUISECONTROLSYSTEM << {speed}.
```

Model - System Sensitivities (*under Adverse Conditions*)


```
||SPEEDHIGH = CRUISECONTROLSYSTEM << {speed}.
```

```
Progress violation for actions:
      {accelerator, brake, engineOff, engineOn,
       off, on, resume, setThrottle, zoom}
Trace to terminal set of states:
      engineOn
Cycle in terminal set:
      speed
Actions in terminal set:
      speed
```

Model - System Sensitivities
(under Adverse Conditions)

```
||SPEEDHIGH = CRUISECONTROLSYSTEM << {speed}.
```

```
Progress violation for actions:
      {accelerator, brake, engineOff, engineOn,
       off, on, resume, setThrottle, zoom}
Trace to terminal set of states:
      engineOn
Cycle in terminal set:
      speed
Actions in terminal set:
      speed
```

Indicates that 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

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.

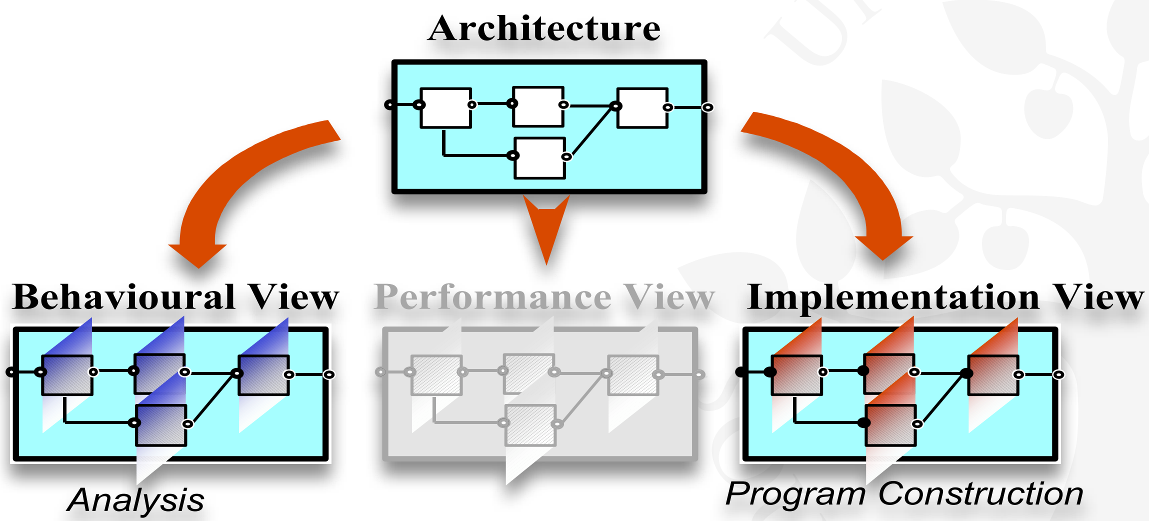
From Models To Implementation

The Central Role Of Design Architecture

Design architecture describes the gross organization and global structure of the system in terms of its constituent components.

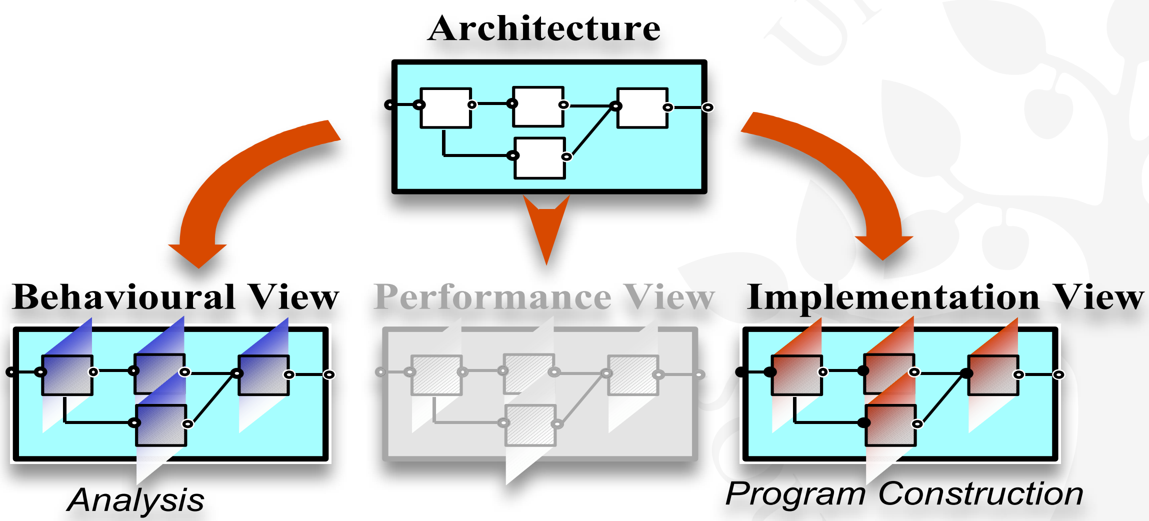
The Central Role Of Design Architecture

Design architecture describes the gross organization and global structure of the system in terms of its constituent components.



The Central Role Of Design Architecture

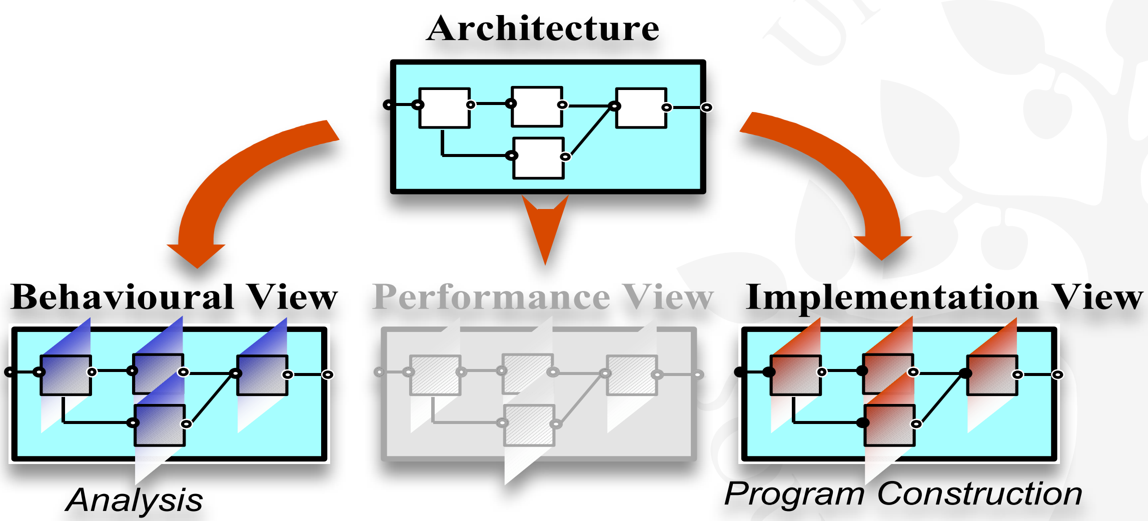
Design architecture describes the gross organization and global structure of the system in terms of its constituent components.



We consider that the implementation should be considered as an **elaborated view** of the basic design architecture.

The Central Role Of Design Architecture

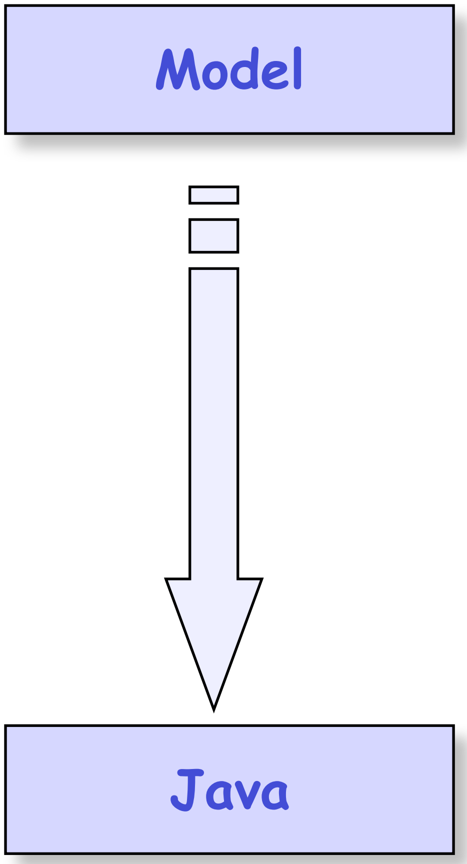
Design architecture describes the gross organization and global structure of the system in terms of its constituent components.



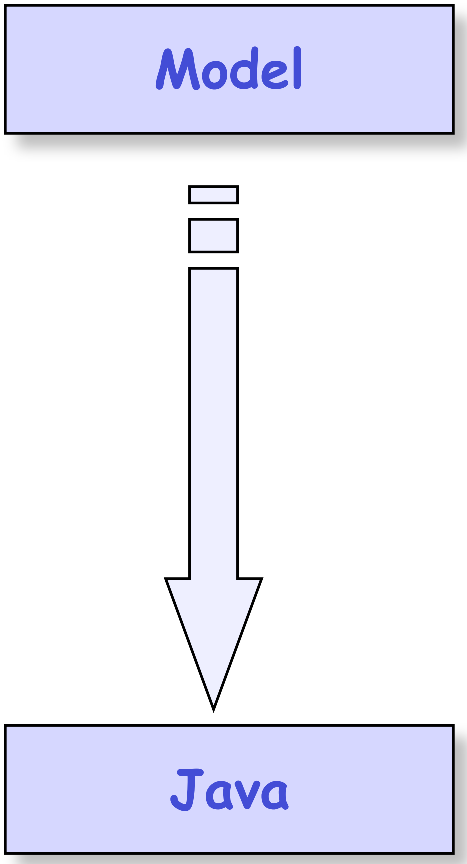
We consider that the implementation should be considered as an **elaborated view** of the basic design architecture.

$S = M_0 \approx M_1 \approx M_2 \approx \ldots \approx M_\infty = I$ // Incremental model refinement

8.2 From Models To Implementations

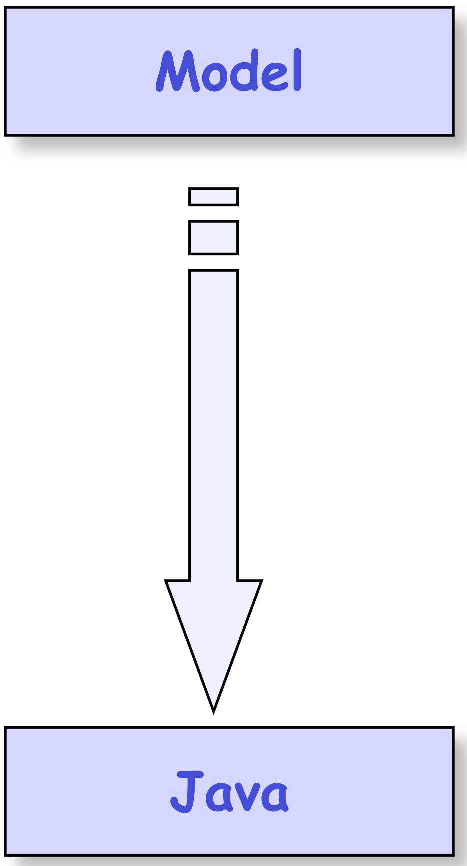


8.2 From Models To Implementations



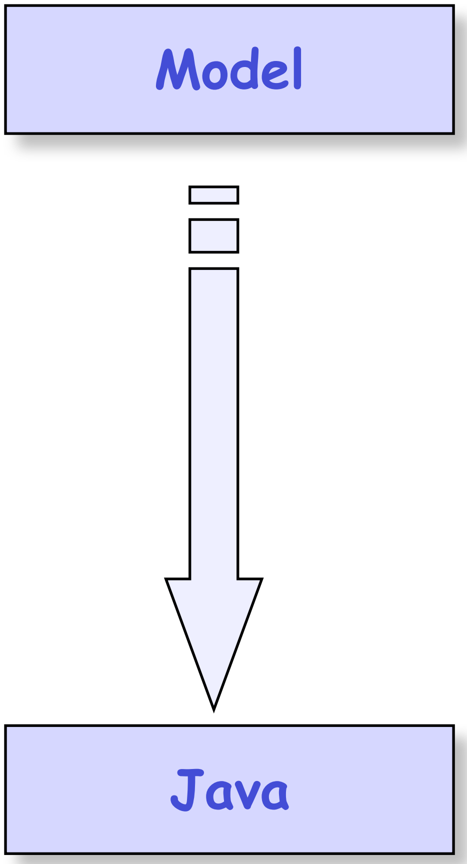
- identify the main active entities
  - to be implemented as threads

8.2 From Models To Implementations



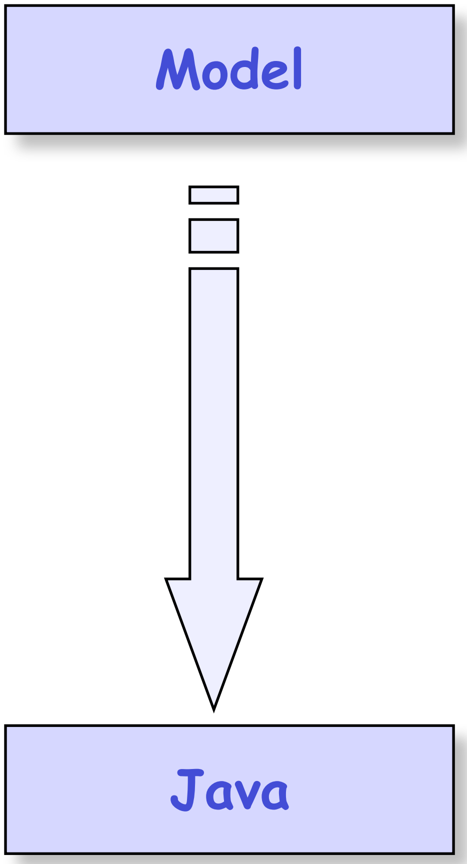
- identify the main active entities
  - to be implemented as threads
- identify the main (shared) passive entities
  - to be implemented as monitors

8.2 From Models To Implementations



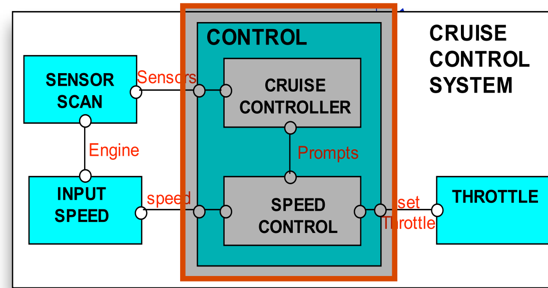
- identify the main active entities
  - to be implemented as threads
- identify the main (shared) passive entities
  - to be implemented as monitors
- (identify the interactive display environment
  - to be implemented as associated classes)

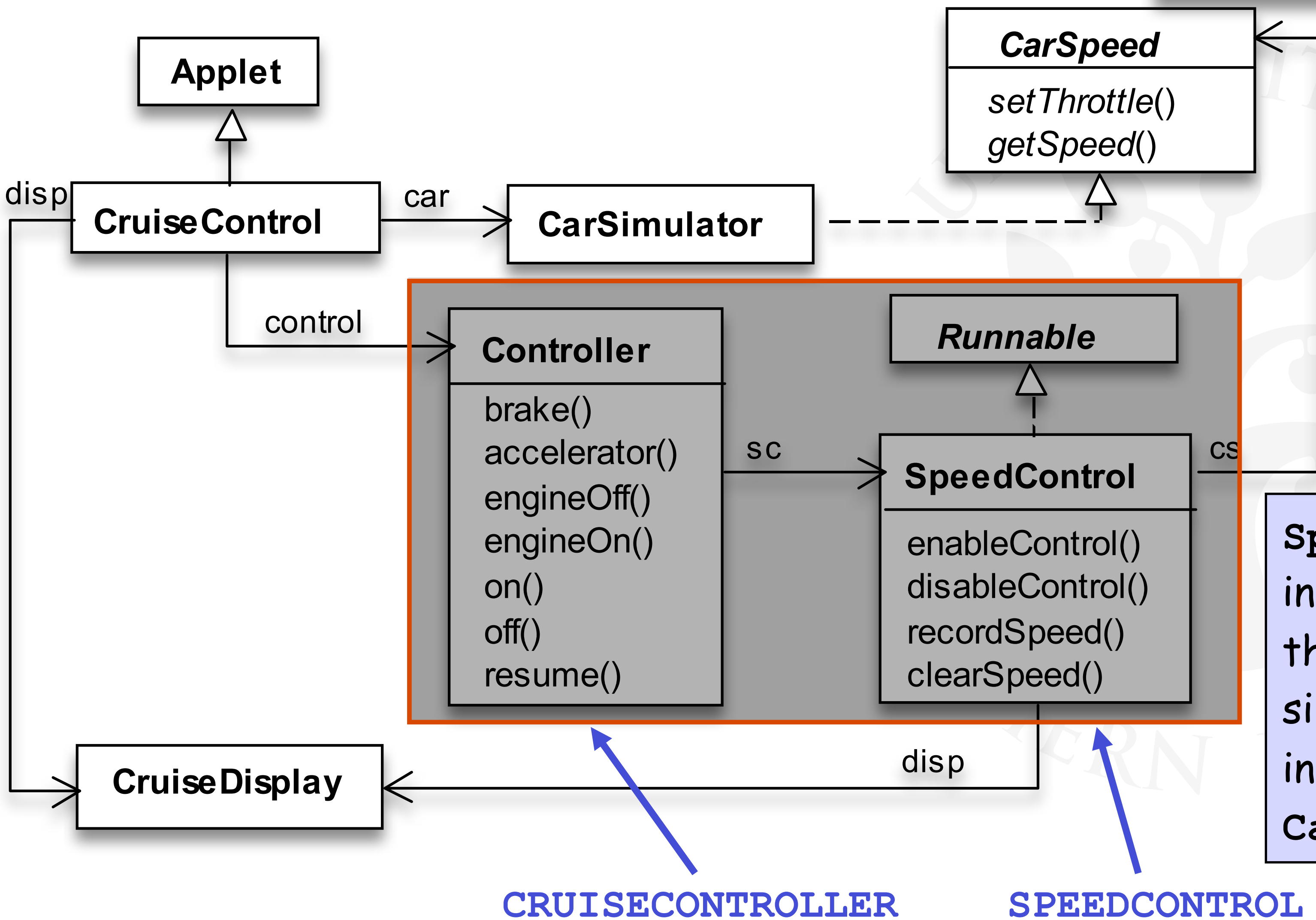
8.2 From Models To Implementations



- 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 (UML) class diagram
  - to be implemented

Cruise Control System - Class Diagram





SpeedControl interacts with the car simulation via interface **CarSpeed**.

Class State-Controller

```
CRUISECONTROLLER = INACTIVE,
INACTIVE =(engineOn  -> clearSpeed            -> ACTIVE),
ACTIVE   =(engineOff                          -> INACTIVE
          |on-> recordSpeed-> enableControl -> CRUISING),
CRUISING =(engineOff       -> disableControl-> INACTIVE
          |{off,brake,accelerator}
                          ->disableControl  -> STANDBY
          |on-> recordSpeed-> enableControl -> CRUISING),
STANDBY  =(engineOff                          -> INACTIVE
          |resume          -> enableControl -> CRUISING
          |on-> recordSpeed-> enableControl -> CRUISING).
```

```java
class StateController {
    final static int INACTIVE = 0,
      ACTIVE = 1, CRUISING = 2, STANDBY = 3; // controller states

    protected int state = INACTIVE;  // initial state
    protected SpeedControl sc;

    StateController(CarSpeed cs, CruiseDisplay disp) {
        sc = new SpeedControl(cs, disp);
    }
    synchronized void brake() {
        if (state == CRUISING)
            { sc.disableControl(); state = STANDBY; }
    }
    synchronized void accelerator() {
        if (state == CRUISING)
            { sc.disableControl(); state = STANDBY; }
    }
    synchronized void engineOff() {
        if (state != INACTIVE) {
            if (state == CRUISING) sc.disableControl();
            state = INACTIVE;
        }
    }
    ...
```

Class State-Controller

```
CRUISECONTROLLER = INACTIVE,
INACTIVE  =(engineOn  -> clearSpeed            -> ACTIVE),
ACTIVE    =(engineOff                          -> INACTIVE
            |on-> recordSpeed-> enableControl  -> CRUISING),
CRUISING  =(engineOff      -> disableControl-> INACTIVE
            |{off,brake,accelerator}
                          ->disableControl  -> STANDBY
            |on-> recordSpeed-> enableControl  -> CRUISING),
STANDBY   =(engineOff                          -> INACTIVE
            |resume        -> enableControl  -> CRUISING
            |on-> recordSpeed-> enableControl  -> CRUISING).
```

```
class StateController {
    final static int INACTIVE = 0,
      ACTIVE = 1, CRUISING = 2, STANDBY = 3; // controller states

    protected int state = INACTIVE;  // initial state
    protected SpeedControl sc;

    StateController(CarSpeed cs, CruiseDisplay disp) {
        sc = new SpeedControl(cs, disp);
    }
    synchronized void brake() {
        if (state == CRUISING)
            { sc.disableControl(); state = STANDBY; }
    }
    synchronized void accelerator() {
        if (state == CRUISING)
            { sc.disableControl(); state = STANDBY; }
    }
    synchronized void engineOff() {
        if (state != INACTIVE) {
            if (state == CRUISING) sc.disableControl();
            state = INACTIVE;
        }
    }
    ...
```

Controller is a passive entity (it reacts to events) and thus implemented as a **monitor**

Class State-Controller

```
CRUISECONTROLLER = INACTIVE,
INACTIVE  =(engineOn  -> clearSpeed          -> ACTIVE),
ACTIVE    =(engineOff                        -> INACTIVE
           |on-> recordSpeed-> enableControl -> CRUISING),
CRUISING  =(engineOff       -> disableControl-> INACTIVE
           |{off,brake,accelerator}
                            ->disableControl -> STANDBY
           |on-> recordSpeed-> enableControl -> CRUISING),
STANDBY   =(engineOff                        -> INACTIVE
           |resume          -> enableControl -> CRUISING
           |on-> recordSpeed-> enableControl -> CRUISING).
```

```java
class StateController {
    final static int INACTIVE = 0,
      ACTIVE = 1, CRUISING = 2, STANDBY = 3; // controller states
```

`Controller` is a passive entity (it reacts to events) and thus implemented as a **monitor**

Class State-Controller

```
CRUISECONTROLLER = INACTIVE,
INACTIVE  =(engineOn  -> clearSpeed            -> ACTIVE),
ACTIVE    =(engineOff                          -> INACTIVE
           |on-> recordSpeed-> enableControl -> CRUISING),
CRUISING  =(engineOff        -> disableControl-> INACTIVE
           |{off,brake,accelerator}
                             ->disableControl -> STANDBY
           |on-> recordSpeed-> enableControl -> CRUISING),
STANDBY   =(engineOff                          -> INACTIVE
           |resume           -> enableControl -> CRUISING
           |on-> recordSpeed-> enableControl -> CRUISING).
```

```java
class StateController {
    final static int INACTIVE = 0,
      ACTIVE = 1, CRUISING = 2, STANDBY = 3; // controller states

    protected int state = INACTIVE;  // initial state
    protected SpeedControl sc;
```

Controller is a passive entity (it reacts to events) and thus implemented as a **monitor**

Class State-Controller

```
CRUISECONTROLLER = INACTIVE,
INACTIVE  =(engineOn  -> clearSpeed           -> ACTIVE),
ACTIVE    =(engineOff                         -> INACTIVE
           |on-> recordSpeed-> enableControl  -> CRUISING),
CRUISING  =(engineOff       -> disableControl-> INACTIVE
           |{off,brake,accelerator}
                           ->disableControl  -> STANDBY
           |on-> recordSpeed-> enableControl  -> CRUISING),
STANDBY   =(engineOff                         -> INACTIVE
           |resume          -> enableControl  -> CRUISING
           |on-> recordSpeed-> enableControl  -> CRUISING).
```

```java
class StateController {
    final static int INACTIVE = 0,
      ACTIVE = 1, CRUISING = 2, STANDBY = 3; // controller states

    protected int state = INACTIVE;  // initial state
    protected SpeedControl sc;

    StateController(CarSpeed cs, CruiseDisplay disp) {
        sc = new SpeedControl(cs, disp);
    }
```

Controller is a passive entity (it reacts to events) and thus implemented as a **monitor**

Class State-Controller

```
CRUISECONTROLLER = INACTIVE,
INACTIVE  =(engineOn  -> clearSpeed            -> ACTIVE),
ACTIVE    =(engineOff                          -> INACTIVE
           |on-> recordSpeed-> enableControl -> CRUISING),
CRUISING  =(engineOff       -> disableControl-> INACTIVE
           |{off,brake,accelerator}
                            ->disableControl -> STANDBY
           |on-> recordSpeed-> enableControl -> CRUISING),
STANDBY   =(engineOff                          -> INACTIVE
           |resume          -> enableControl -> CRUISING
           |on-> recordSpeed-> enableControl -> CRUISING).
```

```
class StateController {
    final static int INACTIVE = 0,
      ACTIVE = 1, CRUISING = 2, STANDBY = 3; // controller states

    protected int state = INACTIVE;  // initial state
    protected SpeedControl sc;

    StateController(CarSpeed cs, CruiseDisplay disp) {
        sc = new SpeedControl(cs, disp);
    }
    synchronized void brake() {
        if (state == CRUISING)
            { sc.disableControl(); state = STANDBY; }
    }
```

Controller is a passive entity (it reacts to events) and thus implemented as a **monitor**

Class State-Controller

```
CRUISECONTROLLER = INACTIVE,
INACTIVE =(engineOn  -> clearSpeed            -> ACTIVE),
ACTIVE   =(engineOff                          -> INACTIVE
          |on-> recordSpeed-> enableControl  -> CRUISING),
CRUISING =(engineOff        -> disableControl-> INACTIVE
          |{off,brake,accelerator}
                            ->disableControl -> STANDBY
          |on-> recordSpeed-> enableControl  -> CRUISING),
STANDBY  =(engineOff                          -> INACTIVE
          |resume           -> enableControl -> CRUISING
          |on-> recordSpeed-> enableControl  -> CRUISING).
```

```java
class StateController {
    final static int INACTIVE = 0,
      ACTIVE = 1, CRUISING = 2, STANDBY = 3; // controller states

    protected int state = INACTIVE;  // initial state
    protected SpeedControl sc;

    StateController(CarSpeed cs, CruiseDisplay disp) {
        sc = new SpeedControl(cs, disp);
    }
    synchronized void brake() {
        if (state == CRUISING)
            { sc.disableControl(); state = STANDBY; }
    }
    synchronized void accelerator() {
        if (state == CRUISING)
            { sc.disableControl(); state = STANDBY; }
    }
```

Controller is a passive entity (it reacts to events) and thus implemented as a **monitor**

Class State-Controller

```
CRUISECONTROLLER = INACTIVE,
INACTIVE  =(engineOn  -> clearSpeed          -> ACTIVE),
ACTIVE    =(engineOff                        -> INACTIVE
           |on-> recordSpeed-> enableControl -> CRUISING),
CRUISING  =(engineOff     -> disableControl-> INACTIVE
           |{off,brake,accelerator}
                          ->disableControl  -> STANDBY
           |on-> recordSpeed-> enableControl -> CRUISING),
STANDBY   =(engineOff                        -> INACTIVE
           |resume        -> enableControl  -> CRUISING
           |on-> recordSpeed-> enableControl -> CRUISING).
```

```java
class StateController {
    final static int INACTIVE = 0,
      ACTIVE = 1, CRUISING = 2, STANDBY = 3; // controller states

    protected int state = INACTIVE;   // initial state
    protected SpeedControl sc;

    StateController(CarSpeed cs, CruiseDisplay disp) {
        sc = new SpeedControl(cs, disp);
    }
    synchronized void brake() {
        if (state == CRUISING)
            { sc.disableControl(); state = STANDBY; }
    }
    synchronized void accelerator() {
        if (state == CRUISING)
            { sc.disableControl(); state = STANDBY; }
    }
    synchronized void engineOff() {
        if (state != INACTIVE) {
            if (state == CRUISING) sc.disableControl();
            state = INACTIVE;
        }
    }
    ...
```

Controller is a passive entity (it reacts to events) and thus implemented as a **monitor**

Class State-`Controller`

```
CRUISECONTROLLER = INACTIVE,
INACTIVE =(engineOn  -> clearSpeed          -> ACTIVE),
ACTIVE   =(engineOff                        -> INACTIVE
          |on-> recordSpeed-> enableControl -> CRUISING),
CRUISING =(engineOff        -> disableControl-> INACTIVE
          |{off,brake,accelerator}
                            ->disableControl -> STANDBY
          |on-> recordSpeed-> enableControl -> CRUISING),
STANDBY  =(engineOff                        -> INACTIVE
          |resume           -> enableControl -> CRUISING
          |on-> recordSpeed-> enableControl -> CRUISING).
```

Class State-Controller

```
CRUISECONTROLLER = INACTIVE,
INACTIVE  =(engineOn  -> clearSpeed          -> ACTIVE),
ACTIVE    =(engineOff                        -> INACTIVE
           |on-> recordSpeed-> enableControl -> CRUISING),
CRUISING  =(engineOff       -> disableControl-> INACTIVE
           |{off,brake,accelerator}
                            ->disableControl -> STANDBY
           |on-> recordSpeed-> enableControl -> CRUISING),
STANDBY   =(engineOff                        -> INACTIVE
           |resume          -> enableControl -> CRUISING
           |on-> recordSpeed-> enableControl -> CRUISING).
```

```
...
synchronized void engineOn() {
    if (state == INACTIVE) {
        sc.clearSpeed(); state = ACTIVE;
    }
}
```

Class State-Controller

```
CRUISECONTROLLER = INACTIVE,
INACTIVE =(engineOn  -> clearSpeed          -> ACTIVE),
ACTIVE   =(engineOff                        -> INACTIVE
          |on-> recordSpeed-> enableControl -> CRUISING),
CRUISING =(engineOff       -> disableControl-> INACTIVE
          |{off,brake,accelerator}
                           ->disableControl -> STANDBY
          |on-> recordSpeed-> enableControl -> CRUISING),
STANDBY  =(engineOff                        -> INACTIVE
          |resume          -> enableControl -> CRUISING
          |on-> recordSpeed-> enableControl -> CRUISING).
```

```
...
synchronized void engineOn() {
    if (state == INACTIVE) {
        sc.clearSpeed(); state = ACTIVE;
    }
}
synchronized void on() {
    if (state != INACTIVE) {
        sc.recordSpeed();
        sc.enableControl(); state = CRUISING;
    }
}
```

Class State-Controller

```
CRUISECONTROLLER = INACTIVE,
INACTIVE  =(engineOn  -> clearSpeed          -> ACTIVE),
ACTIVE    =(engineOff                        -> INACTIVE
            |on-> recordSpeed-> enableControl -> CRUISING),
CRUISING  =(engineOff       -> disableControl-> INACTIVE
            |{off,brake,accelerator}
                           ->disableControl  -> STANDBY
            |on-> recordSpeed-> enableControl -> CRUISING),
STANDBY   =(engineOff                        -> INACTIVE
            |resume         -> enableControl -> CRUISING
            |on-> recordSpeed-> enableControl -> CRUISING).
```

```
...
synchronized void engineOn() {
    if (state == INACTIVE) {
        sc.clearSpeed(); state = ACTIVE;
    }
}
synchronized void on() {
    if (state != INACTIVE) {
        sc.recordSpeed();
        sc.enableControl(); state = CRUISING;
    }
}
synchronized void off() {
    if (state == CRUISING) {
        sc.disableControl(); state = STANDBY;
    }
}
```

Class State-Controller

```
CRUISECONTROLLER = INACTIVE,
INACTIVE =(engineOn  -> clearSpeed          -> ACTIVE),
ACTIVE   =(engineOff                        -> INACTIVE
          |on-> recordSpeed-> enableControl -> CRUISING),
CRUISING =(engineOff        -> disableControl-> INACTIVE
          |{off,brake,accelerator}
                            ->disableControl -> STANDBY
          |on-> recordSpeed-> enableControl -> CRUISING),
STANDBY  =(engineOff                        -> INACTIVE
          |resume           -> enableControl -> CRUISING
          |on-> recordSpeed-> enableControl -> CRUISING).
```

```
    ...
    synchronized void engineOn() {
        if (state == INACTIVE) {
            sc.clearSpeed(); state = ACTIVE;
        }
    }
    synchronized void on() {
        if (state != INACTIVE) {
            sc.recordSpeed();
            sc.enableControl(); state = CRUISING;
        }
    }
    synchronized void off() {
        if (state == CRUISING) {
            sc.disableControl(); state = STANDBY;
        }
    }
    synchronized void resume() {
        if (state == STANDBY) {
            sc.enableControl(); state = CRUISING;
        }
    }
}
```

Class `SpeedControl`

```
SPEEDCONTROL = DISABLED,
DISABLED =({speed,clearSpeed,recordSpeed} -> DISABLED
          |enableControl                  -> ENABLED),
ENABLED = (speed      -> setThrottle      -> ENABLED
          |{recordSpeed,enableControl}    -> ENABLED
          |disableControl                 -> DISABLED).
```

Class SpeedControl

```
SPEEDCONTROL = DISABLED,
DISABLED =({speed,clearSpeed,recordSpeed} -> DISABLED
          |enableControl                  -> ENABLED),
ENABLED = (speed      -> setThrottle      -> ENABLED
          |{recordSpeed,enableControl}    -> ENABLED
          |disableControl                 -> DISABLED).
```

```java
class SpeedControl implements Runnable {
```

Class `SpeedControl`

```
SPEEDCONTROL = DISABLED,
DISABLED =({speed,clearSpeed,recordSpeed} -> DISABLED
          |enableControl                  -> ENABLED),
ENABLED = (speed      -> setThrottle      -> ENABLED
          |{recordSpeed,enableControl}    -> ENABLED
          |disableControl                 -> DISABLED).
```

```
class SpeedControl implements Runnable {
```

`SpeedControl` is an active entity; when enabled, a **new thread** is created (which periodically obtains car speed and sets the throttle).

Class SpeedControl

```
SPEEDCONTROL = DISABLED,
DISABLED =({speed,clearSpeed,recordSpeed} -> DISABLED
          |enableControl                  -> ENABLED),
ENABLED = (speed      -> setThrottle      -> ENABLED
          |{recordSpeed,enableControl}    -> ENABLED
          |disableControl                 -> DISABLED).
```

```java
class SpeedControl implements Runnable {
    final static int DISABLED = 0; // speed control states
    final static int ENABLED  = 1;
    protected int state = DISABLED;  // initial state
```

`SpeedControl` is an active entity; when enabled, a **new thread** is created (which periodically obtains car speed and sets the throttle).

Class SpeedControl

```
SPEEDCONTROL = DISABLED,
DISABLED =({speed,clearSpeed,recordSpeed} -> DISABLED
          |enableControl                  -> ENABLED),
ENABLED = (speed      -> setThrottle      -> ENABLED
          |{recordSpeed,enableControl}    -> ENABLED
          |disableControl                 -> DISABLED).
```

```java
class SpeedControl implements Runnable {
    final static int DISABLED = 0; // speed control states
    final static int ENABLED  = 1;
    protected int state = DISABLED;  // initial state
    protected int set_speed = 0;     // initial speed setting
```

SpeedControl is an active entity; when enabled, a **new thread** is created (which periodically obtains car speed and sets the throttle).

Class SpeedControl

```
SPEEDCONTROL = DISABLED,
DISABLED =({speed,clearSpeed,recordSpeed} -> DISABLED
          |enableControl                  -> ENABLED),
ENABLED = (speed      -> setThrottle      -> ENABLED
          |{recordSpeed,enableControl}    -> ENABLED
          |disableControl                 -> DISABLED).
```

```java
class SpeedControl implements Runnable {
    final static int DISABLED = 0; // speed control states
    final static int ENABLED  = 1;
    protected int state = DISABLED;  // initial state
    protected int set_speed = 0;     // initial speed setting
    protected Thread sc;
    protected CarSpeed cs;     // interface to the car (simulator)
    protected CruiseDisplay disp;
```

SpeedControl is an active entity; when enabled, a **new thread** is created (which periodically obtains car speed and sets the throttle).

Class SpeedControl

```
SPEEDCONTROL = DISABLED,
DISABLED =({speed,clearSpeed,recordSpeed} -> DISABLED
          |enableControl                  -> ENABLED),
ENABLED = (speed     -> setThrottle       -> ENABLED
          |{recordSpeed,enableControl}    -> ENABLED
          |disableControl                 -> DISABLED).
```

```java
class SpeedControl implements Runnable {
    final static int DISABLED = 0; // speed control states
    final static int ENABLED  = 1;
    protected int state = DISABLED;  // initial state
    protected int set_speed = 0;     // initial speed setting
    protected Thread sc;
    protected CarSpeed cs;     // interface to the car (simulator)
    protected CruiseDisplay disp;

    SpeedControl(CarSpeed c, CruiseDisplay d){
        this.cs = c; this.disp = d; d.disable(); d.record(0);
    }
```

`SpeedControl` is an active entity; when enabled, a **new thread** is created (which periodically obtains car speed and sets the throttle).

Class SpeedControl

```
SPEEDCONTROL = DISABLED,
DISABLED =({speed,clearSpeed,recordSpeed} -> DISABLED
          |enableControl                  -> ENABLED),
ENABLED = (speed      -> setThrottle      -> ENABLED
          |{recordSpeed,enableControl}    -> ENABLED
          |disableControl                 -> DISABLED).
```

```
class SpeedControl implements Runnable {
    final static int DISABLED = 0; // speed control states
    final static int ENABLED  = 1;
    protected int state = DISABLED;  // initial state
    protected int set_speed = 0;     // initial speed setting
    protected Thread sc;
    protected CarSpeed cs;      // interface to the car (simulator)
    protected CruiseDisplay disp;

    SpeedControl(CarSpeed c, CruiseDisplay d){
        this.cs = c; this.disp = d; d.disable(); d.record(0);
    }

    synchronized void recordSpeed() {
        set_speed = cs.getSpeed();
        disp.record(set_speed);
    }
```

SpeedControl is an active entity; when enabled, a **new thread** is created (which periodically obtains car speed and sets the throttle).

Class SpeedControl

```
SPEEDCONTROL = DISABLED,
DISABLED =({speed,clearSpeed,recordSpeed} -> DISABLED
          |enableControl                  -> ENABLED),
ENABLED = (speed     -> setThrottle       -> ENABLED
          |{recordSpeed,enableControl}    -> ENABLED
          |disableControl                 -> DISABLED).
```

```
class SpeedControl implements Runnable {
    final static int DISABLED = 0; // speed control states
    final static int ENABLED  = 1;
    protected int state = DISABLED;  // initial state
    protected int set_speed = 0;     // initial speed setting
    protected Thread sc;
    protected CarSpeed cs;     // interface to the car (simulator)
    protected CruiseDisplay disp;

    SpeedControl(CarSpeed c, CruiseDisplay d){
        this.cs = c; this.disp = d; d.disable(); d.record(0);
    }

    synchronized void recordSpeed() {
        set_speed = cs.getSpeed();
        disp.record(set_speed);
    }

    synchronized void clearSpeed() {
        if (state == DISABLED) {
            set_speed = 0;
            disp.record(set_speed);
        }
    }
```

SpeedControl is an active entity; when enabled, a **new thread** is created (which periodically obtains car speed and sets the throttle).

Class `SpeedControl`

```
SPEEDCONTROL = DISABLED,
DISABLED =({speed,clearSpeed,recordSpeed} -> DISABLED
          |enableControl                  -> ENABLED),
ENABLED = (speed      -> setThrottle      -> ENABLED
          |{recordSpeed,enableControl}    -> ENABLED
          |disableControl                 -> DISABLED).
```

Class SpeedControl

```
SPEEDCONTROL = DISABLED,
DISABLED =({speed,clearSpeed,recordSpeed} -> DISABLED
          |enableControl                  -> ENABLED),
ENABLED = (speed     -> setThrottle       -> ENABLED
          |{recordSpeed,enableControl}    -> ENABLED
          |disableControl                 -> DISABLED).
```

```
...

synchronized void enableControl() {
    if (state == DISABLED) {
        disp.enable();
        sc = new Thread(this);
        sc.start();
        state = ENABLED;
    }
}
```

Class SpeedControl

```
SPEEDCONTROL = DISABLED,
DISABLED =({speed,clearSpeed,recordSpeed} -> DISABLED
          |enableControl                  -> ENABLED),
ENABLED = (speed      -> setThrottle      -> ENABLED
          |{recordSpeed,enableControl}    -> ENABLED
          |disableControl                 -> DISABLED).
```

```
...

synchronized void enableControl() {
    if (state == DISABLED) {
        disp.enable();
        sc = new Thread(this);
        sc.start();
        state = ENABLED;
    }
}

synchronized void disableControl() {
    if (state == ENABLED) {
        disp.disable();
        state = DISABLED;
    }
}

...
```

Class SpeedControl

```
SPEEDCONTROL = DISABLED,
DISABLED =({speed,clearSpeed,recordSpeed} -> DISABLED
          |enableControl                   -> ENABLED),
ENABLED = (speed      -> setThrottle       -> ENABLED
          |{recordSpeed,enableControl}     -> ENABLED
          |disableControl                  -> DISABLED).
```

Class SpeedControl

```
SPEEDCONTROL = DISABLED,
DISABLED =({speed,clearSpeed,recordSpeed} -> DISABLED
          |enableControl                   -> ENABLED),
ENABLED = (speed      -> setThrottle       -> ENABLED
          |{recordSpeed,enableControl}     -> ENABLED
          |disableControl                  -> DISABLED).
```

```
    public void run() { // the speed controller thread
```

Class SpeedControl

```
SPEEDCONTROL = DISABLED,
DISABLED =({speed,clearSpeed,recordSpeed} -> DISABLED
          |enableControl                  -> ENABLED),
ENABLED = (speed     -> setThrottle       -> ENABLED
          |{recordSpeed,enableControl}    -> ENABLED
          |disableControl                 -> DISABLED).
```

```
public void run() { // the speed controller thread
    try {



    } catch (InterruptedException _) {}
```

Class SpeedControl

```
SPEEDCONTROL = DISABLED,
DISABLED =({speed,clearSpeed,recordSpeed} -> DISABLED
          |enableControl                  -> ENABLED),
ENABLED = (speed      -> setThrottle      -> ENABLED
          |{recordSpeed,enableControl}    -> ENABLED
          |disableControl                 -> DISABLED).
```

```
public void run() { // the speed controller thread
    try {
        while (state == ENABLED) {



    } catch (InterruptedException _) {}
```

Class SpeedControl

```
SPEEDCONTROL = DISABLED,
DISABLED =({speed,clearSpeed,recordSpeed} -> DISABLED
          |enableControl                  -> ENABLED),
ENABLED = (speed      -> setThrottle      -> ENABLED
          |{recordSpeed,enableControl}     -> ENABLED
          |disableControl                  -> DISABLED).
```

```
public void run() { // the speed controller thread
    try {
        while (state == ENABLED) {
            double s = speed();


    } catch (InterruptedException _) {}
```

Class SpeedControl

```
SPEEDCONTROL = DISABLED,
DISABLED =({speed,clearSpeed,recordSpeed} -> DISABLED
          |enableControl                  -> ENABLED),
ENABLED = (speed      -> setThrottle      -> ENABLED
          |{recordSpeed,enableControl}    -> ENABLED
          |disableControl                 -> DISABLED).
```

```
public void run() { // the speed controller thread
    try {
        while (state == ENABLED) {
            double s = speed();
            setThrottle(s);


    } catch (InterruptedException _) {}
```

Class SpeedControl

```
SPEEDCONTROL = DISABLED,
DISABLED =({speed,clearSpeed,recordSpeed} -> DISABLED
          |enableControl                   -> ENABLED),
ENABLED = (speed      -> setThrottle       -> ENABLED
          |{recordSpeed,enableControl}     -> ENABLED
          |disableControl                  -> DISABLED).
```

```
    public void run() { // the speed controller thread
        try {
            while (state == ENABLED) {
                double s = speed();
                setThrottle(s);
                Thread.sleep(500);

        } catch (InterruptedException _) {}
```

Class SpeedControl

```
SPEEDCONTROL = DISABLED,
DISABLED =({speed,clearSpeed,recordSpeed} -> DISABLED
          |enableControl                  -> ENABLED),
ENABLED = (speed     -> setThrottle       -> ENABLED
          |{recordSpeed,enableControl}    -> ENABLED
          |disableControl                 -> DISABLED).
```

```
    public void run() { // the speed controller thread
        try {
            while (state == ENABLED) {
                double s = speed();
                setThrottle(s);
                Thread.sleep(500);
            }
        } catch (InterruptedException _) {}
```

Class `SpeedControl`

```
SPEEDCONTROL = DISABLED,
DISABLED =({speed,clearSpeed,recordSpeed} -> DISABLED
          |enableControl                  -> ENABLED),
ENABLED = (speed      -> setThrottle      -> ENABLED
          |{recordSpeed,enableControl}    -> ENABLED
          |disableControl                 -> DISABLED).
```

```java
public void run() { // the speed controller thread
    try {
        while (state == ENABLED) {
            double s = speed();
            setThrottle(s);
            Thread.sleep(500);
        }
    } catch (InterruptedException _) {}
    sc = null; // throw away SpeedController thread
}
```

Class SpeedControl

```
SPEEDCONTROL = DISABLED,
DISABLED =({speed,clearSpeed,recordSpeed} -> DISABLED
          |enableControl                  -> ENABLED),
ENABLED = (speed      -> setThrottle     -> ENABLED
          |{recordSpeed,enableControl}   -> ENABLED
          |disableControl                -> DISABLED).
```

```
public void run() { // the speed controller thread
    try {
        while (state == ENABLED) {
            double s = speed();
            setThrottle(s);
            Thread.sleep(500);
        }
    } catch (InterruptedException _) {}
    sc = null; // throw away SpeedController thread
}

synchronized private double speed() {
    return ...cs.getSpeed()...;
}
```

Class SpeedControl

```
SPEEDCONTROL = DISABLED,
DISABLED =({speed,clearSpeed,recordSpeed} -> DISABLED
          |enableControl                   -> ENABLED),
ENABLED = (speed      -> setThrottle       -> ENABLED
          |{recordSpeed,enableControl}     -> ENABLED
          |disableControl                  -> DISABLED).
```

```
public void run() { // the speed controller thread
    try {
        while (state == ENABLED) {
            double s = speed();
            setThrottle(s);
            Thread.sleep(500);
        }
    } catch (InterruptedException _) {}
    sc = null; // throw away SpeedController thread
}

synchronized private double speed() {
    return ...cs.getSpeed()...;
}
synchronized private void setThrottle(double throttle) {
    cs.setThrottle(...throttle...);
}
```

Class SpeedControl

```
SPEEDCONTROL = DISABLED,
DISABLED =({speed,clearSpeed,recordSpeed} -> DISABLED
          |enableControl                  -> ENABLED),
ENABLED = (speed     -> setThrottle        -> ENABLED
          |{recordSpeed,enableControl}    -> ENABLED
          |disableControl                 -> DISABLED).
```

```
public void run() { // the speed controller thread
    try {
        while (state == ENABLED) {
            double s = speed();
            setThrottle(s);
            Thread.sleep(500);
        }
    } catch (InterruptedException _) {}
    sc = null; // throw away SpeedController thread
}

synchronized private double speed() {
    return ...cs.getSpeed()...;
}
synchronized private void setThrottle(double throttle) {
    cs.setThrottle(...throttle...);
}
```

SpeedControl is an example of a class that combines both

- synchronized methods (to update local vars); and
- a thread.

Summary: Model-Based Design

Summary: Model-Based Design

Concepts:

design process:

- from requirements to **models**
- from **models** to implementations

Summary: Model-Based Design

Concepts:

design process:

- from requirements to **models**
- from **models** to implementations

Models:

check properties of interest:

- safety of the appropriate (sub-)system
- progress of the overall system

Summary: Model-Based Design

Concepts:

design process:

- from requirements to **models**
- from **models** to implementations

Models:

check properties of interest:

- safety of the appropriate (sub-)system
- progress of the overall system

Practice:

model "interpretation":

- to infer actual system behavior

active threads and passive monitors

Summary: Model-Based Design

Concepts:

design process:
- from requirements to **models**
- from **models** to implementations

Models:

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

Practice:

model "interpretation":
- to infer actual system behavior

active threads and passive monitors

Aim: rigorous design process.