Example: Model Train Controller

Purposes of example:
Follow a design through several levels of abstraction.
Gain experience with UML.

Text: Section 1.4
Model train setup

- rcvr
- motor
- power supply
- console
- header
- address
- command
- ECC
Requirements

- Console controls up to 8 trains on 1 track.
- Throttle has at least 63 levels.
- Inertia control adjusts responsiveness with at least 8 levels.
- Emergency stop button.
- Error detection scheme on messages.
  - Ignore erroneous messages
## Requirements form

<table>
<thead>
<tr>
<th>name</th>
<th>model train controller</th>
</tr>
</thead>
<tbody>
<tr>
<td>purpose</td>
<td>control speed of &lt;= 8 model trains</td>
</tr>
<tr>
<td>inputs</td>
<td>throttle, inertia, emergency stop, train #</td>
</tr>
<tr>
<td>outputs</td>
<td>train control signals</td>
</tr>
<tr>
<td>functions</td>
<td>set engine speed w. inertia; emergency stop</td>
</tr>
<tr>
<td>performance</td>
<td>can update train speed at least 10 times/sec</td>
</tr>
<tr>
<td>manufacturing cost</td>
<td>$50</td>
</tr>
<tr>
<td>power</td>
<td>wall powered</td>
</tr>
<tr>
<td>physical</td>
<td>console comfortable for 2 hands; &lt; 2 lbs.</td>
</tr>
</tbody>
</table>
Conceptual specification

- Before we create a detailed specification, we will make an initial, simplified specification.
  - Gives us practice in specification and UML.
  - Good idea in general to identify potential problems before investing too much effort in detail.
Basic system commands

Command-name parameters

set-speed   speed (positive/negative)
set-inertia inertia-value (non-negative)
estop       none
Typical control sequence

Time

Console always monitoring buttons/knobs

Receiver always "listening"

:console

set-inertia

set-speed

set-speed

estop

set-speed

:train_rcvr
Implemented message classes derived from message class.

- Attributes and operations will be filled in for detailed specification.

- Implemented message classes specify message type by their class.

- May have to add type as parameter to data structure in implementation.
Subsystem collaboration diagram

Shows relationship between console and receiver (ignores role of track): interaction via commands

sequence 1..n: command message type

:console :receiver
System structure modeling

- Some classes define non-computer components.
  - Denote by *name.
- Choose important systems at this point to show basic roles and relationships.

Major subsystem roles

- **Console:**
  - read state of front panel;
  - format messages;
  - transmit messages.

- **Train:**
  - receive message;
  - interpret message;
  - control the train.
- **panel**: describes analog knobs and interface hardware.
- **formatter**: turns knob settings into bit streams.
- **transmitter**: sends data on track.
- **receiver**: digitizes signal from track.
- **controller**: interprets received commands and makes control decisions.
- **motor interface**: generates signals required by motor.
Detailed specification

- We can now fill in the details of the conceptual specification:
  - more classes;
  - behaviors.
- Sketching out the spec first helps us understand the basic relationships in the system.
Train system analog physical object classes

- **knobs***
  - train-knob: integer
  - speed-knob: integer
  - inertia-knob: unsigned-integer
  - emergency-stop: boolean
  - set_knobs()

- **pulser***
  - pulse-width: unsigned-integer
  - direction: boolean

- **sender***
  - send-bit()

- **detector***
  - read-bit() : integer

Motor controlled by pulse width modulation:

```
+ V -
```

Duty cycle:

```
[ ] [ ]
```
Panel and motor interface classes

- **panel** class defines the controls.
  - new-settings() function reads the controls.
- **motor-interface** class defines the motor speed/inertia, held as state.

### Panel
- train-number() : integer
- speed() : integer
- inertia() : integer
- estop() : boolean
- new-settings()

### Motor-interface
- speed: integer
- inertia: integer
Control input cases

- Use a soft panel to show current panel settings for each train.
- Changing train number:
  - must change soft panel settings to reflect current train’s speed, etc.
- Controlling throttle/inertia/estop:
  - read panel, check for changes, perform command.
Transmitter and receiver classes

- **transmitter class** has one method for each type of message sent.
- **receiver class** provides methods to:
  - detect a new message;
  - determine its type;
  - read its parameters (estop has no parameters).

<table>
<thead>
<tr>
<th>Transmitter</th>
<th>Receiver</th>
</tr>
</thead>
<tbody>
<tr>
<td>send-speed(adrs: integer, speed: integer)</td>
<td>current: command</td>
</tr>
<tr>
<td>send-inertia(adrs: integer, val: integer)</td>
<td>new: boolean</td>
</tr>
<tr>
<td>send-estop(adrs: integer)</td>
<td>read-cmd()</td>
</tr>
<tr>
<td>rcv-type(msg-type: command)</td>
<td>new-cmd(): boolean</td>
</tr>
<tr>
<td>rcv-speed(val: integer)</td>
<td>rcv-inertia(val:integer)</td>
</tr>
</tbody>
</table>
Formatter class

- **Formatter class** holds state for each train, setting for current train.
- The **operate()** operation performs the basic formatting task.

<table>
<thead>
<tr>
<th>formatter</th>
</tr>
</thead>
<tbody>
<tr>
<td>current-train: integer</td>
</tr>
<tr>
<td>current-speed[ntrains]: integer</td>
</tr>
<tr>
<td>current-inertia[ntrains]: unsigned-integer</td>
</tr>
<tr>
<td>current-estop[ntrains]: boolean</td>
</tr>
</tbody>
</table>

send-command()
panel-active() : boolean
operate()
Control input sequence diagram

- :knobs
  - change in speed/inertia/estop
  - change in control settings
  - change in train number
  - set-knobs

- :panel
  - read panel
  - panel settings

- :formatter
  - panel settings
  - send-command
  - send-speed, send-inertia, send-estop

- :transmitter
  - panel-active
  - send-speed, send-inertia, send-estop
Formatter operate behavior
(in the formatter class)

- idle
- update-panel()
- panel-active()
- new train number
- send-command()
- other
Panel-active behavior
(in the formatter class)

- `panel*:read-train()`
  - `current-train != train-knob`
  - `T` → `current-train = train-knob`
  - `F` → `changed = true`

- `panel*:read-speed()`
  - `current-speed != throttle`
  - `T` → `current-speed = throttle`
  - `F` → `changed = true`

...
Train controller class

controller

- current-train: integer
- current-speed[ntrains]: integer
- current-direction[ntrains]: boolean
- current-inertia[ntrains]: unsigned-integer

operate()
issue-command()
Setting the speed

- Don’t want to change speed instantaneously.
- Controller should change speed gradually by sending several commands.
wait for a command from receiver

Controller operate behavior

receive-command() -> issue-command()
Sequence diagram for set-speed command

(receiver) -> new-cmd
    (controller) -> cmd-type
        (motor-interface) -> rcv-speed
            (pulser*) <- set-speed
                (pulser*) <- set-pulse
                    (pulser*) <- set-pulse
                        (pulser*) <- set-pulse
                            (pulser*) <- set-pulse
Refined command classes

- **command**
  - type: 3-bits
  - address: 3-bits
  - parity: 1-bit

- **set-speed**
  - type=010
  - value: 7-bits

- **set-inertia**
  - type=001
  - value: 3-bits

- **estop**
  - type=000
Summary

- Separate specification and programming.
  - Small mistakes are easier to fix in the spec.
  - Big mistakes in programming cost a lot of time.

- You can’t completely separate specification and architecture.
  - Make a few tasteful assumptions.