Example: Model Train Controller

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

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

- Console can control 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.
<table>
<thead>
<tr>
<th>Requirement</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>name</strong></td>
<td>model train controller</td>
</tr>
<tr>
<td><strong>purpose</strong></td>
<td>control speed of $\leq 8$ model trains</td>
</tr>
<tr>
<td><strong>inputs</strong></td>
<td>throttle, inertia, emergency stop, train #</td>
</tr>
<tr>
<td><strong>outputs</strong></td>
<td>train control signals</td>
</tr>
<tr>
<td><strong>functions</strong></td>
<td>set engine speed w. inertia; emergency stop</td>
</tr>
<tr>
<td><strong>performance</strong></td>
<td>can update train speed at least 10 times/sec</td>
</tr>
<tr>
<td><strong>manufacturing cost</strong></td>
<td>$50</td>
</tr>
<tr>
<td><strong>power</strong></td>
<td>wall powered</td>
</tr>
<tr>
<td><strong>physical size/weight</strong></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

:console

set-inertia

set-speed

set-speed

estop

set-speed

:train_rcvr

Time
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
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.
Console system classes

- **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 speed control

- Motor controlled by pulse width modulation:

```
+------------------
<p>| V                |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
</table>

duty cycle
```
Train system analog physical object classes

<table>
<thead>
<tr>
<th>knobs*</th>
<th>pulser*</th>
<th>sender*</th>
<th>detector*</th>
</tr>
</thead>
<tbody>
<tr>
<td>train-knob: integer</td>
<td>pulse-width: unsigned-integer</td>
<td>send-bit()</td>
<td>read-bit() : integer</td>
</tr>
<tr>
<td>speed-knob: integer</td>
<td>direction: boolean</td>
<td></td>
<td></td>
</tr>
<tr>
<td>inertia-knob: unsigned-integer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>emergency-stop: boolean</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

set_knobs()
Panel and motor interface classes

- **panel** class defines the controls.
  - new-settings() behavior reads the controls.
- **motor-interface** class defines the motor speed held as state.
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).

### transmitter

- send-speed(adrs: integer, speed: integer)
- send-inertia(adrs: integer, val: integer)
- send-estop(adrs: integer)

### receiver

- current: command
- new: boolean
- read-cmd()
- new-cmd(): boolean
- rcv-type(msg-type: command)
- rcv-speed(val: integer)
- rcv-inertia(val: integer)
**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>
<tr>
<td>send-command()</td>
</tr>
<tr>
<td>panel-active() : boolean</td>
</tr>
<tr>
<td>operate()</td>
</tr>
</tbody>
</table>
Control input sequence diagram

:knobs

change in speed/inertia/estop

control settings

change in train number

set-knobs

:panel

read panel

panel settings

read panel

panel-active

send-command

send-speed, send-inertia, send-estop

:formatter

panel settings

:transmitter

send-command

new-settings

panel settings

read panel
Formatter operate behavior
(in the formatter class)

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

```
panel*:read-train()

current-train != train-knob

panel*:read-speed()

current-speed != throttle
```
Train controller class

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

| operate() |
| issue-command() |
Setting the speed

- Don’t want to change speed instantaneously.
- Controller should change speed gradually by sending several commands.
Controller operate behavior

wait for a command from receiver

receive-command()

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

1. :receiver
   - new-cmd
   - cmd-type
   - rcv-speed

2. :controller
   - set-speed

3. :motor-interface
   - set-pulse
   - set-pulse
   - set-pulse

4. :pulser*
   - set-pulse
   - set-pulse
   - 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.