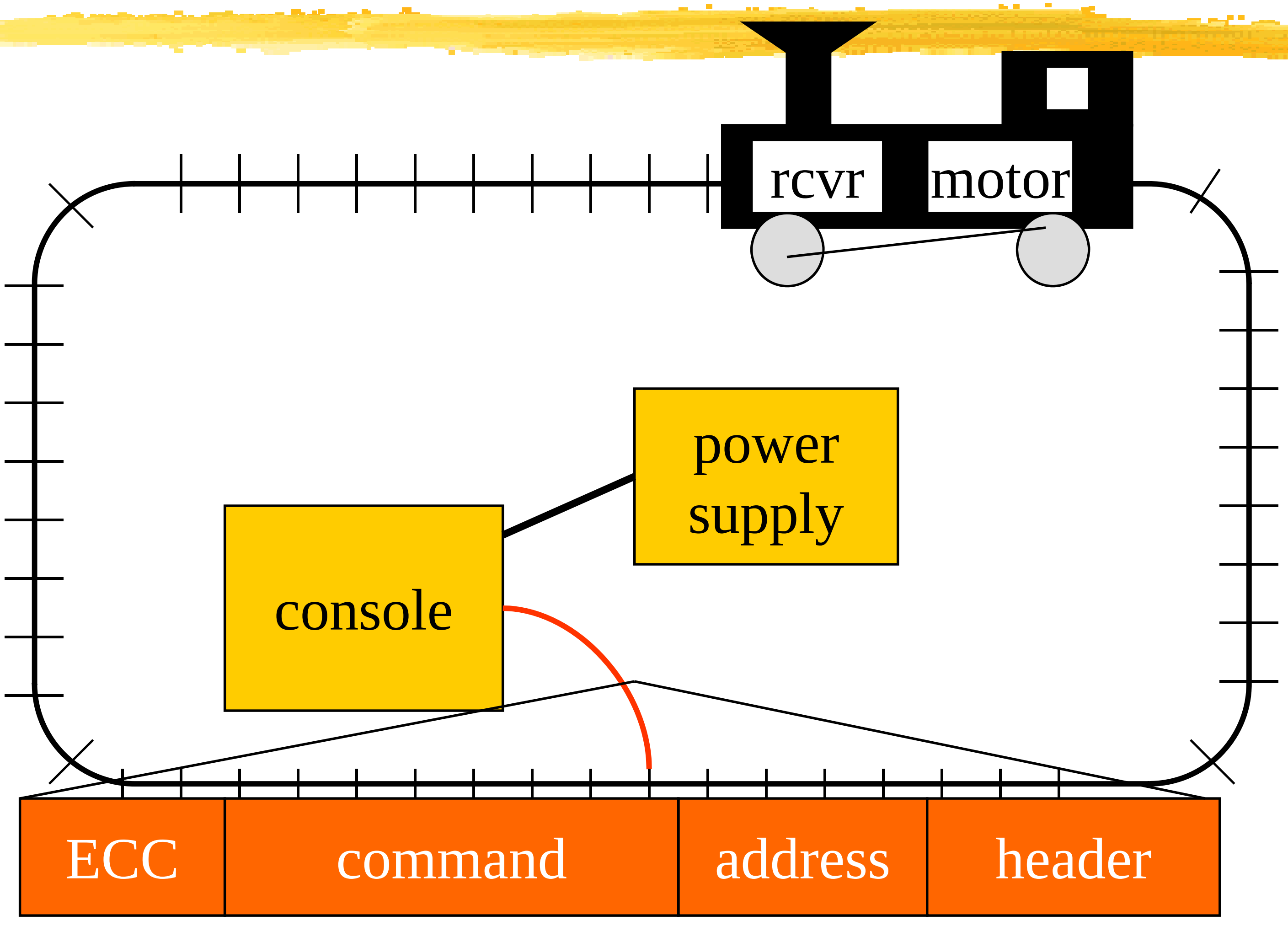
Introduction

⌘Example: model train controller.

Purposes of example

- Follow a design through several levels of abstraction.
- Gain experience with UML.

Model train setup



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.

Requirements form

name	model train controller
purpose	control speed of <= 8 model trains
inputs	throttle, inertia, emergency stop, train #
outputs	train control signals
functions	set engine speed w. inertia; emergency stop
performance	can update train speed at least 10 times/sec
manufacturing cost	$50
power	10 W (wall powered)
physical size/weight	console comfortable for 2 hands; < 2 lbs.

Digital Command Control

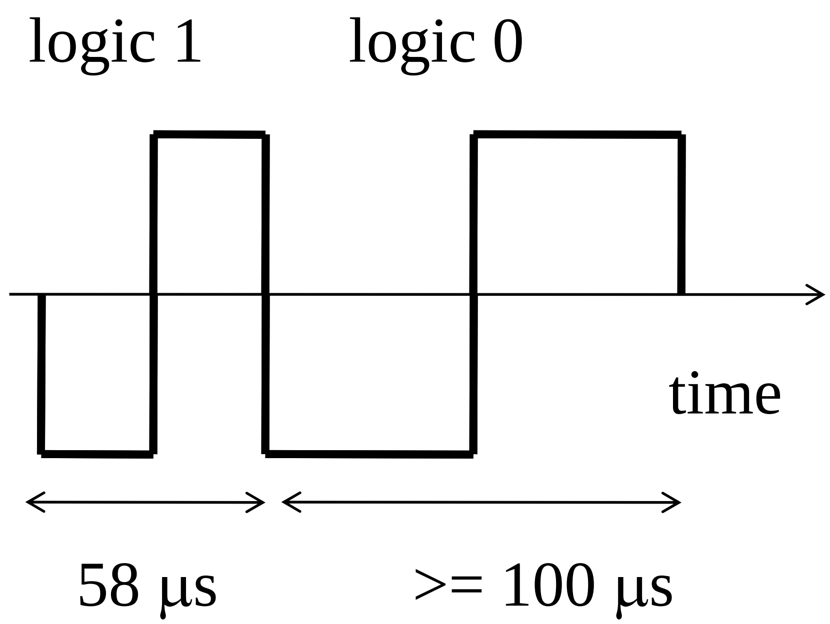
- DCC created by model railroad hobbyists, picked up by industry.
- Defines way in which model trains, controllers communicate.
  - Leaves many system design aspects open, allowing competition.
- This is a simple example of a big trend:
  - Cell phones, digital TV rely on standards.

DCC documents

- Standard S-9.1, DCC Electrical Standard.
  - Defines how bits are encoded on the rails.
- Standard S-9.2, DCC Communication Standard.
  - Defines packet format and semantics.

DCC electrical standard

- Voltage moves around the power supply voltage; adds no DC component.
- 1 is 58 μs, 0 is at least 100 μs.



DCC communication standard

- Basic packet format: PSA(sD)+E.
- P: preamble = 1111111111.
- S: packet start bit = 0.
- A: address data byte.
- s: data byte start bit.
- D: data byte (data payload).
- E: packet end bit = 1.
- A packet include one or more data byte start bit/ data byte combination.

DCC packet types

- A baseline packet: minimum packet that must be accepted by all DCC implementations, which has three data bytes.
  - a address data byte gives receiver address.
  - an instruction data byte gives basic instruction.
  - an error correction data byte gives ECC.

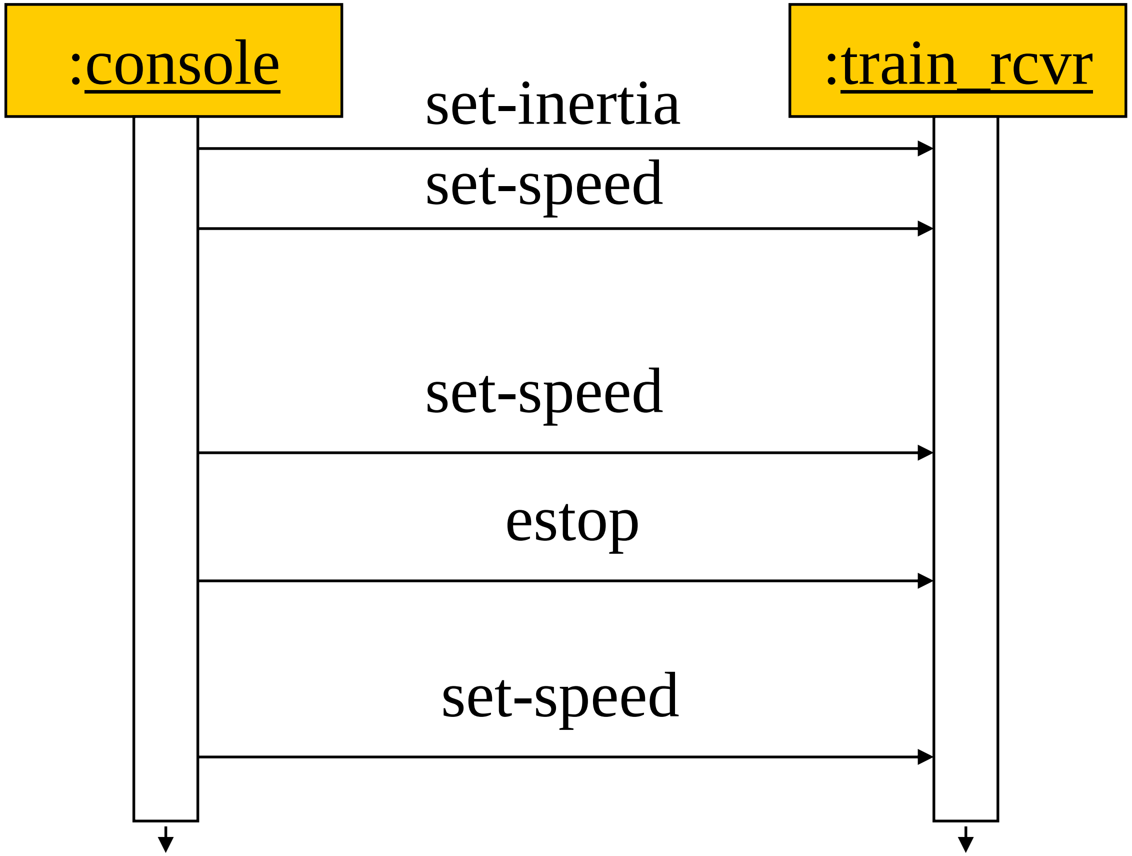
Conceptual specification

- 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.
- Commands and packets may not be generated in a 1-to-1 ratio.

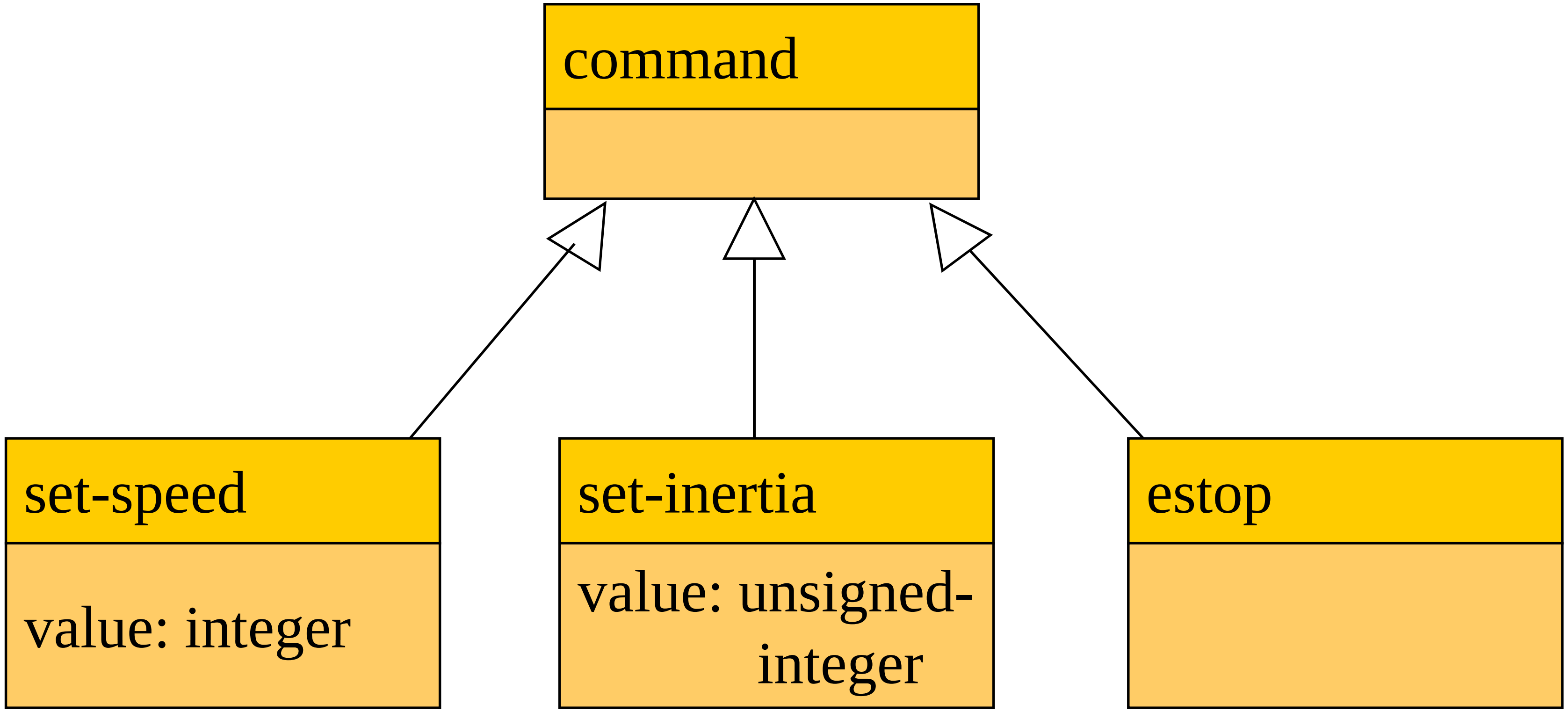
Basic system commands

command name	parameters
set-speed	speed (positive/negative)
set-inertia	inertia-value (non-negative)
estop	none

Typical control sequence



Message classes



Roles of message classes

- 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.

Collaboration diagram

- Interaction diagram
- Shows relationship between console and receiver (ignores role of track):

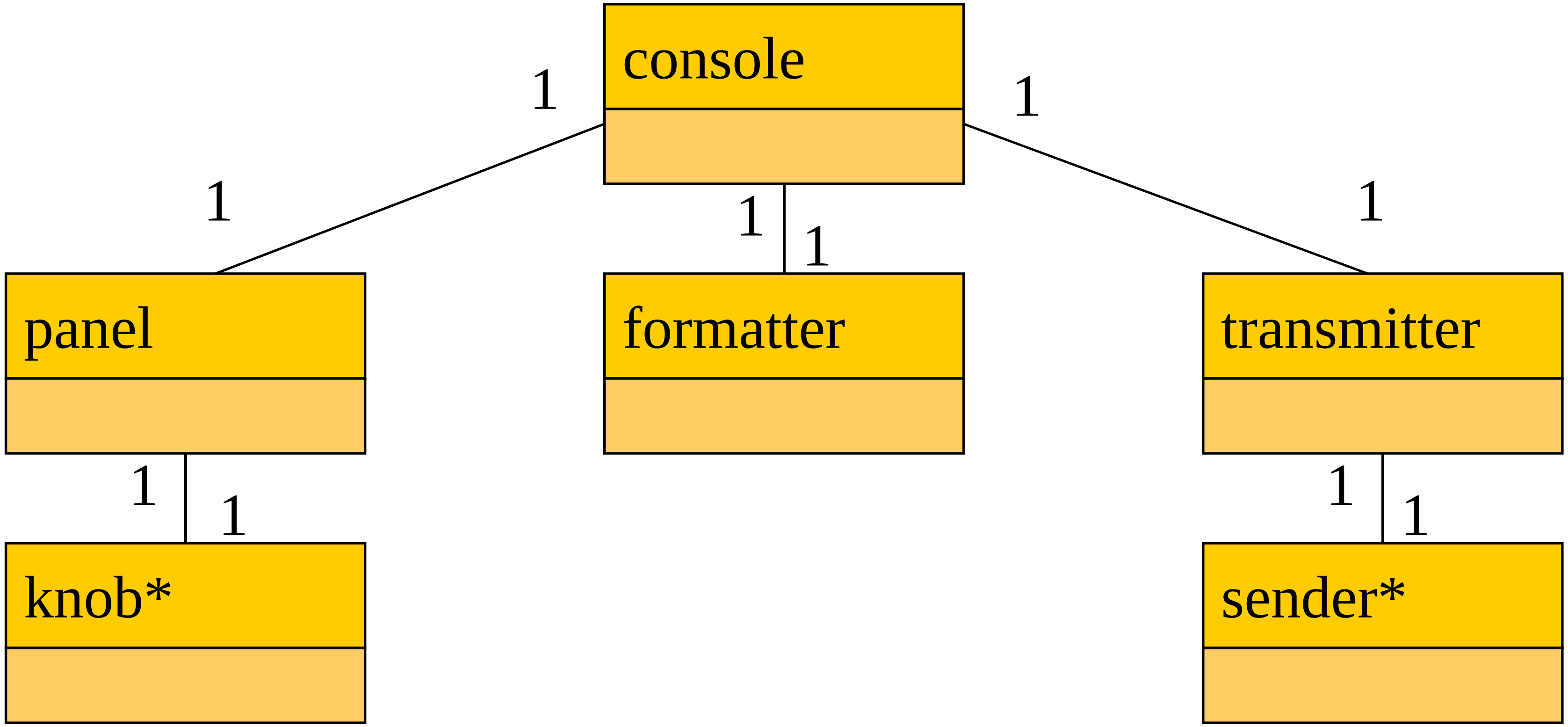
System structure modeling

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

Major subsystem roles

- Console:
  - read state of front panel;
  - format messages;
  - transmit messages.
- Train:
  - receive message;
  - interpret message;
  - control the train.

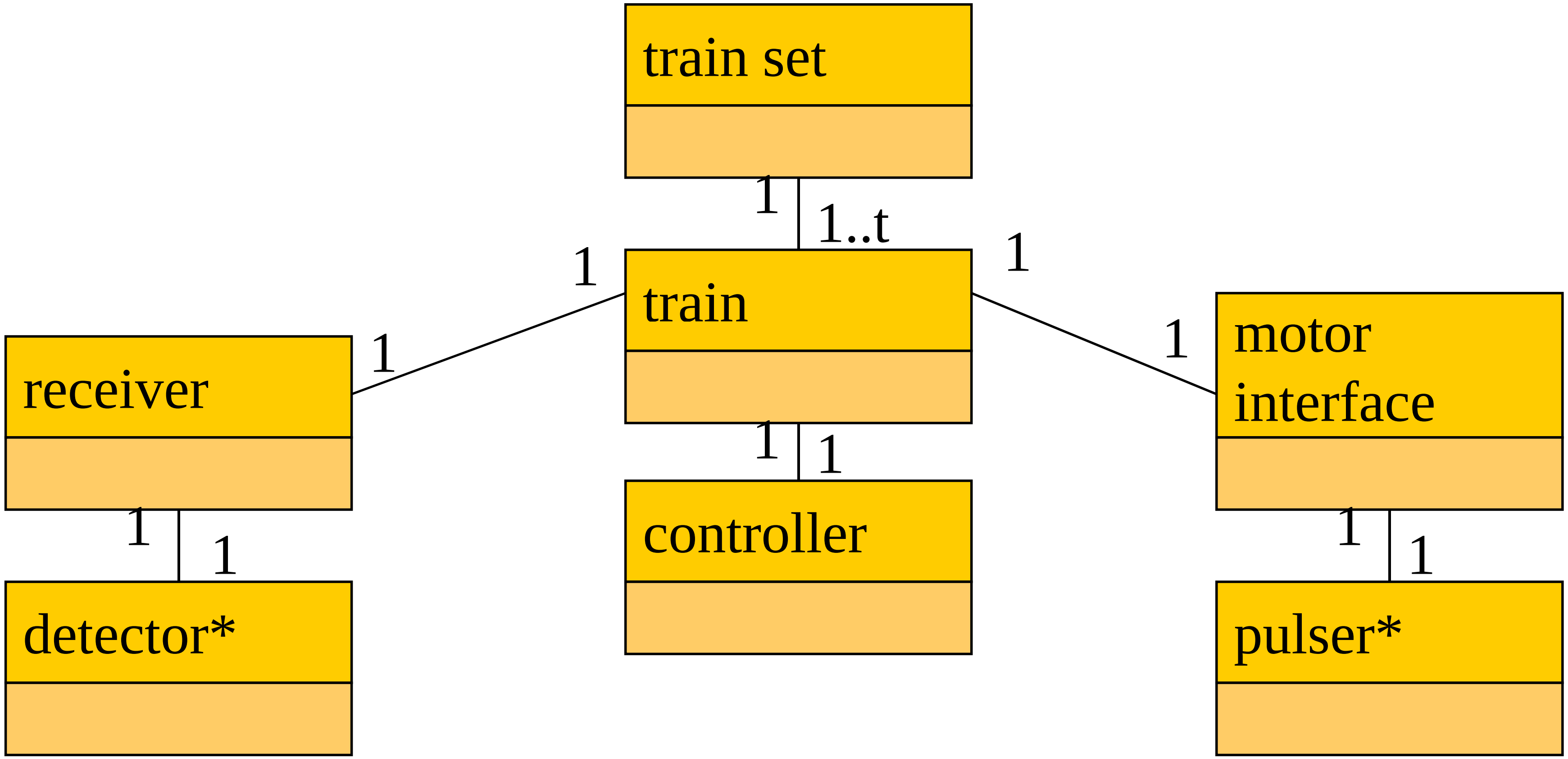
Console system classes



Console class roles

- panel: describes analog knobs and interface hardware.
- formatter: turns knob settings into bit streams.
- transmitter: sends data on track.

Train system classes



Train class roles

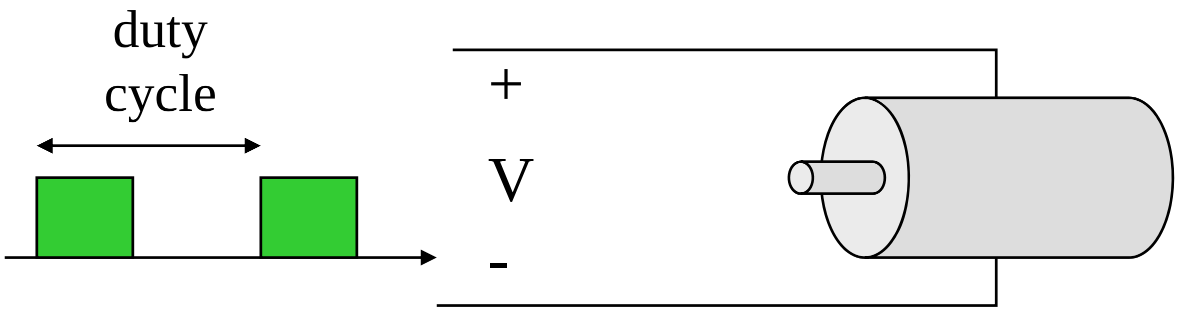
- **receiver**: digitizes signal from track.
- **controller**: interprets received commands and makes control decisions.
- **motor interface**: generates signals required by motor.

Detailed specification

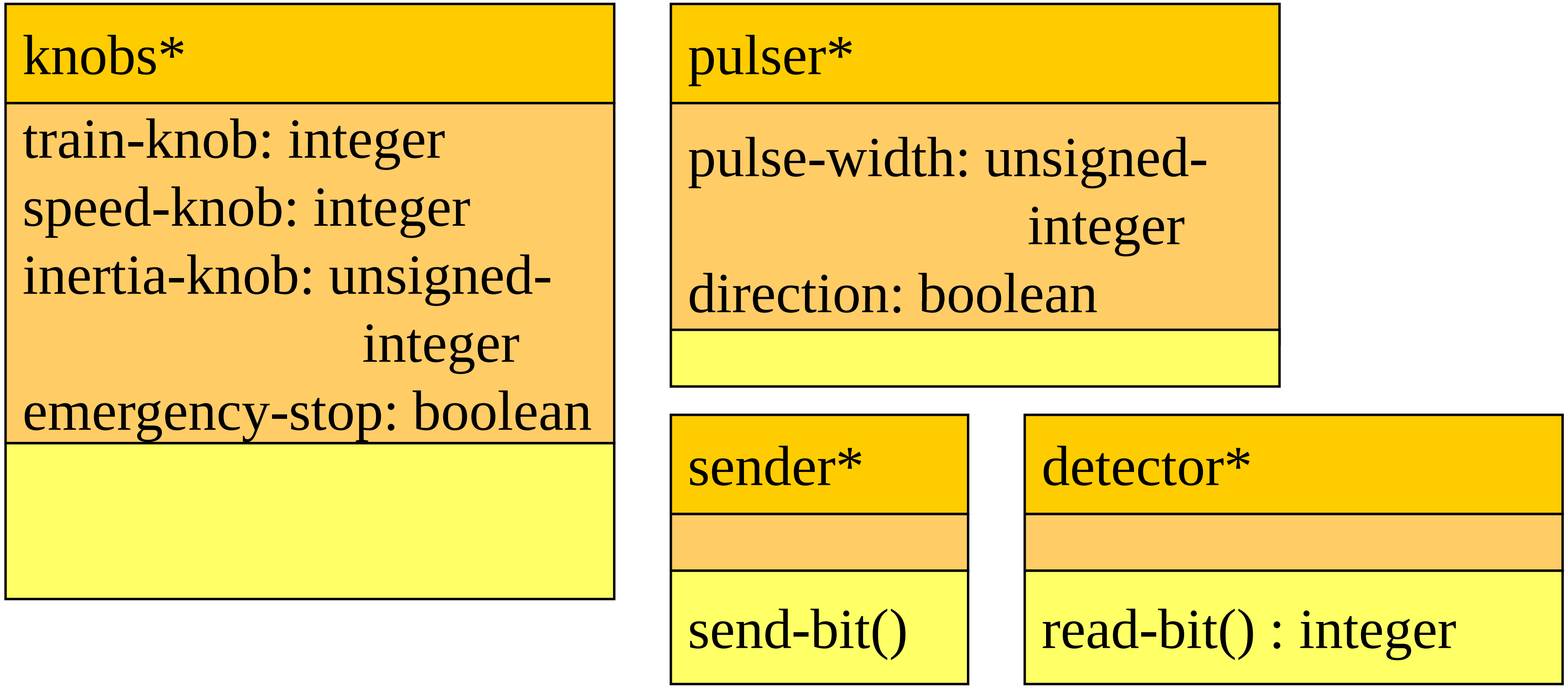
- We can now fill in the details of the conceptual specification:
  - more classes;
  - behaviors.
- Sketching out the spec first helps us understand the basic relationships in the system.

Train speed control

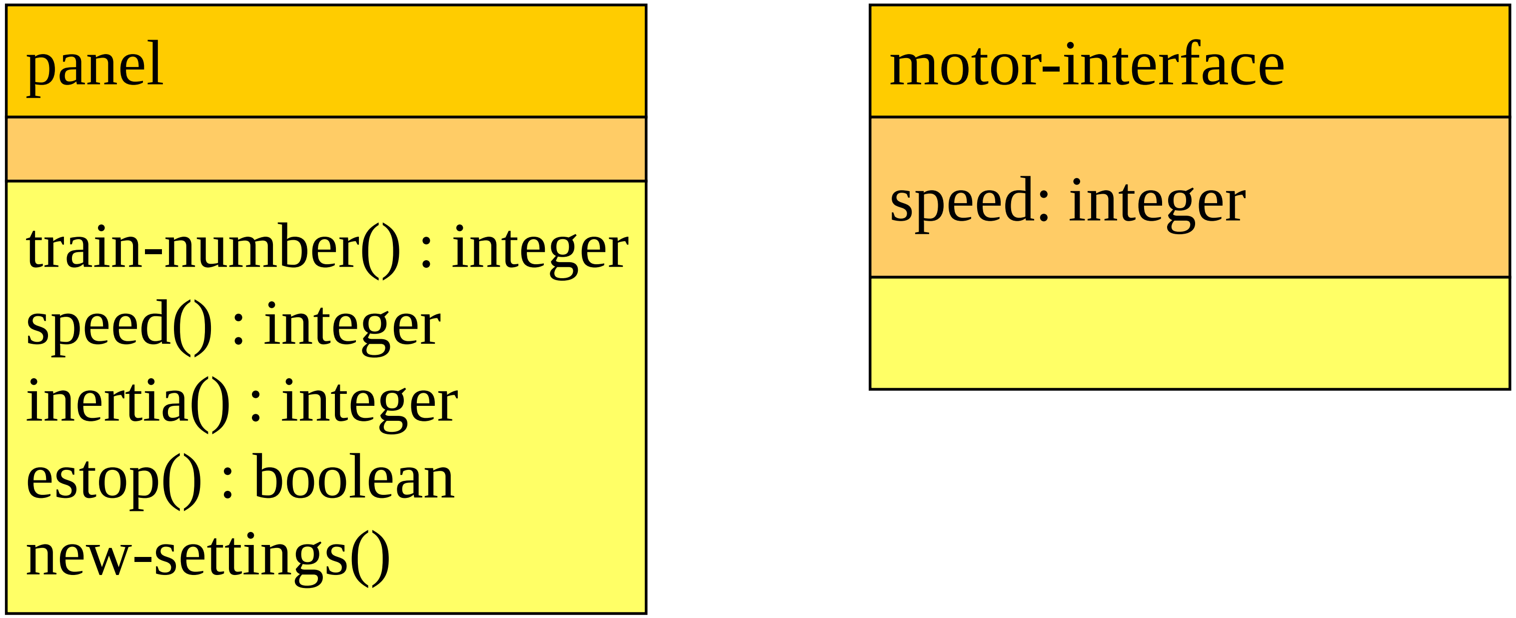
⌘Motor controlled by pulse width modulation:



Console physical object classes



Panel and motor interface classes

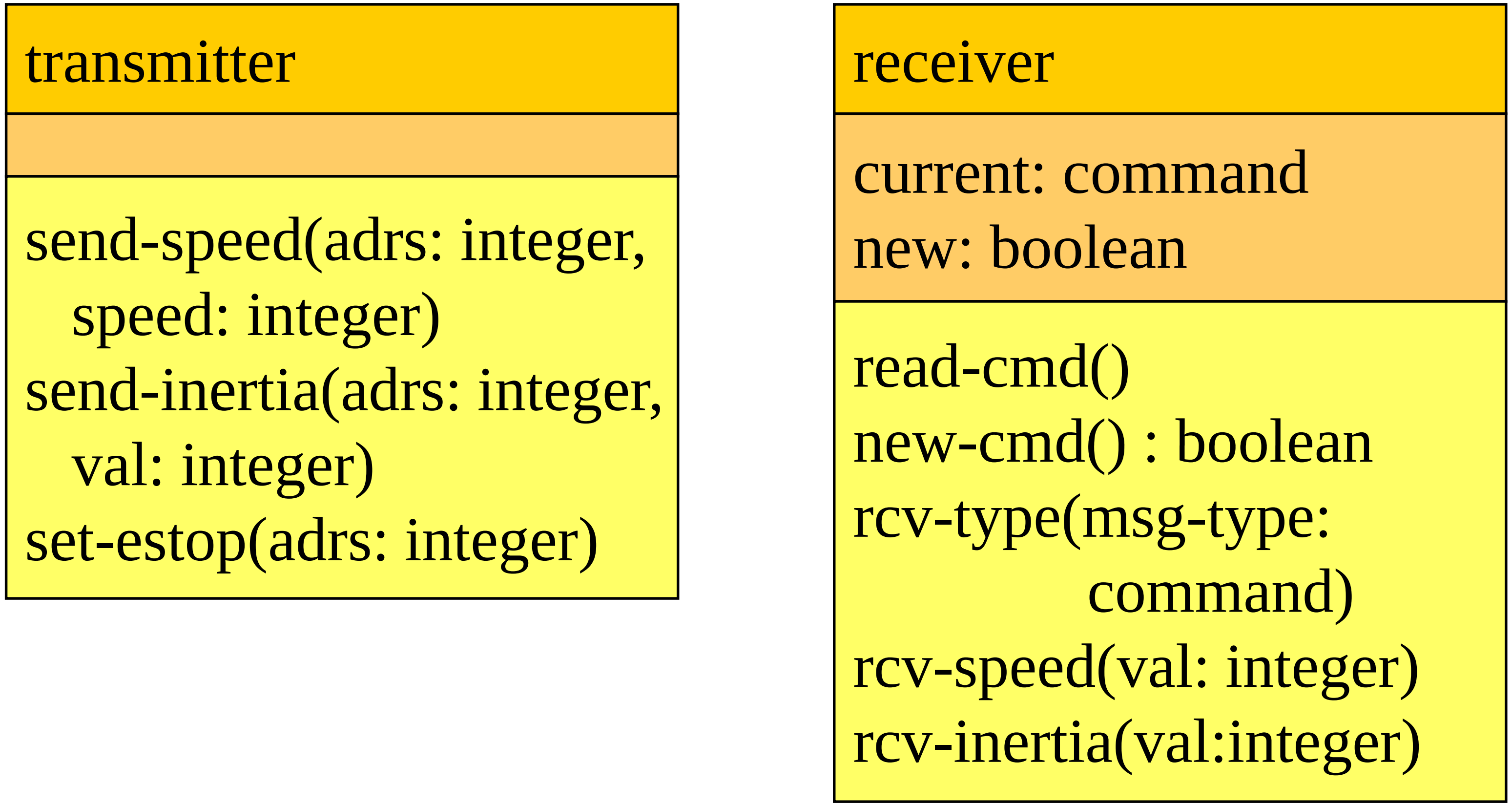


new-settings(): use the set-knobs behavior of the Knobs* class
to read the knobs settings whenever the train number setting is changed

Class descriptions

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

Transmitter and receiver classes



Class descriptions

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

Formatter class

formatter
current-train: integer current-speed[ntrains]: integer current-inertia[ntrains]: unsigned-integer current-estop[ntrains]: boolean
send-command() panel-active() : boolean operate()

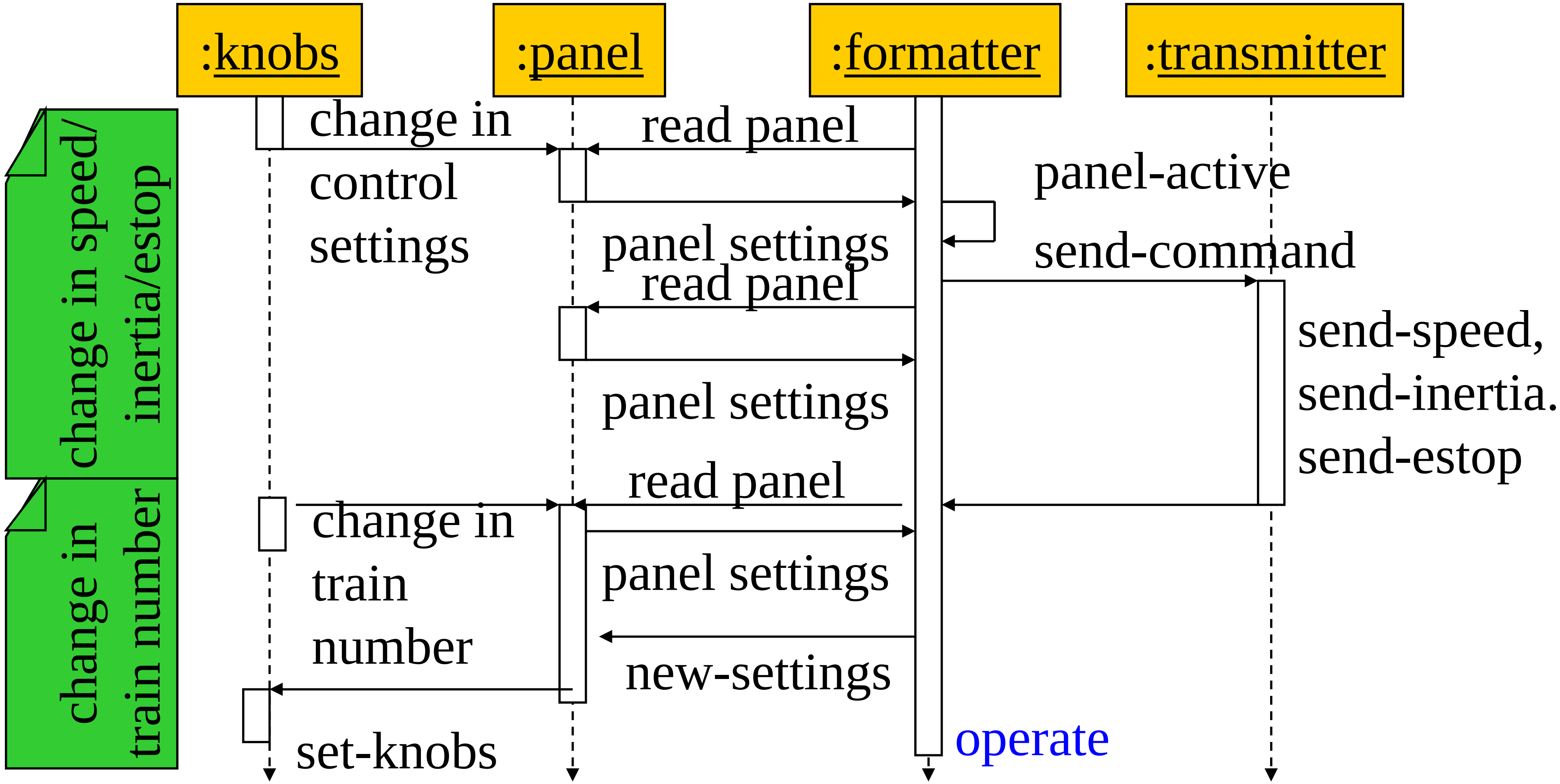
Formatter class description

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

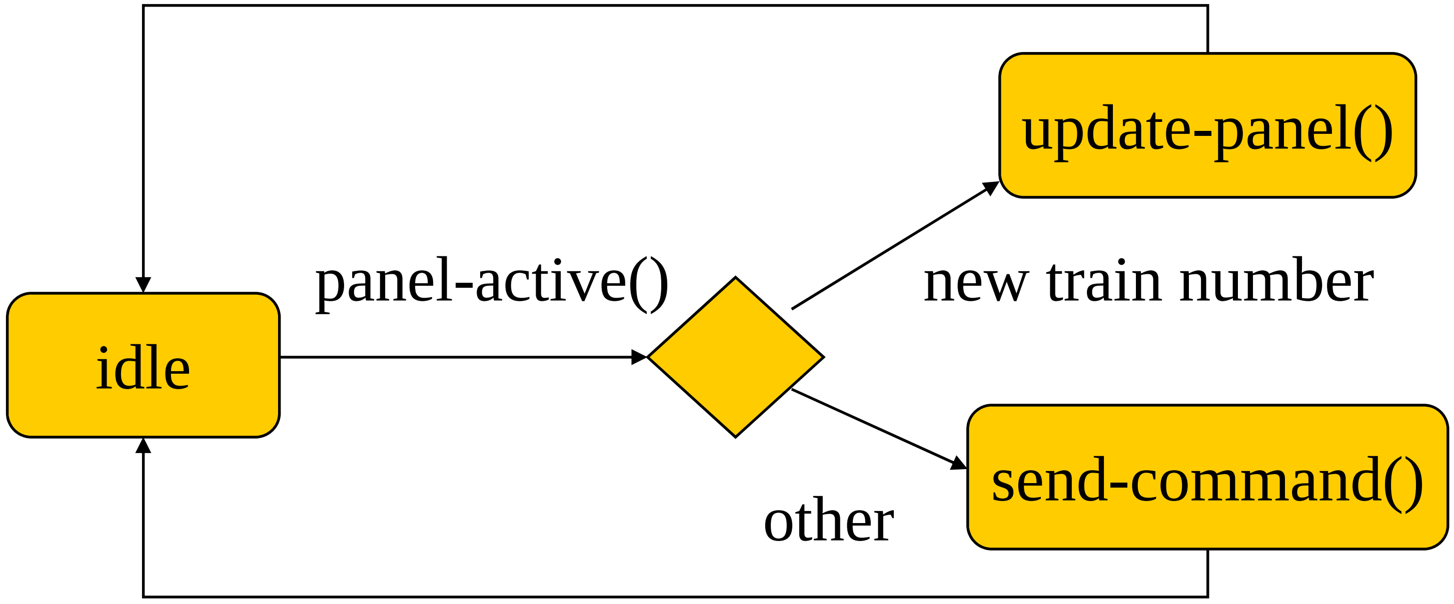
Control input cases

- 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.

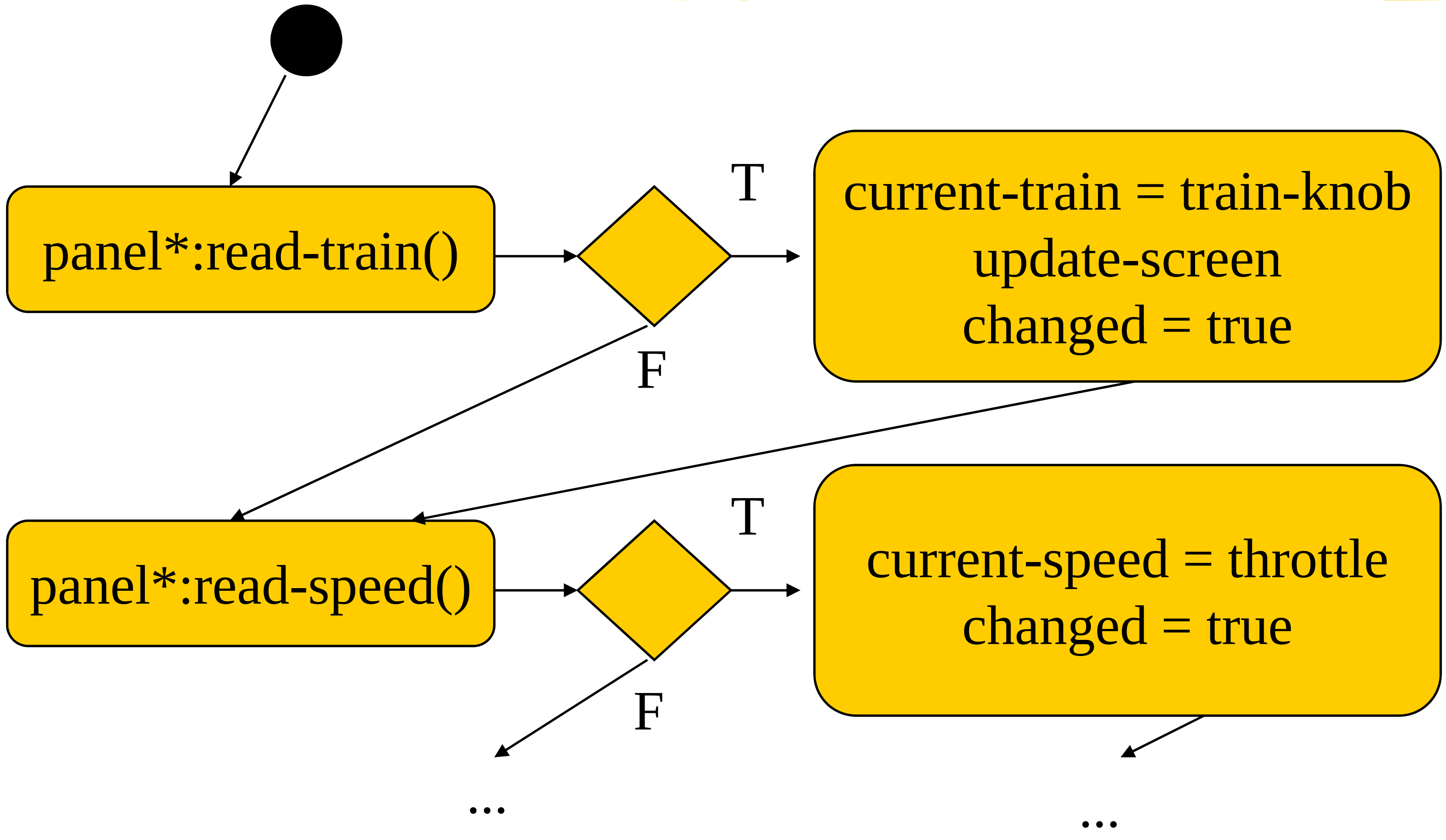
Control input sequence diagram



Formatter operate behavior



Panel-active behavior



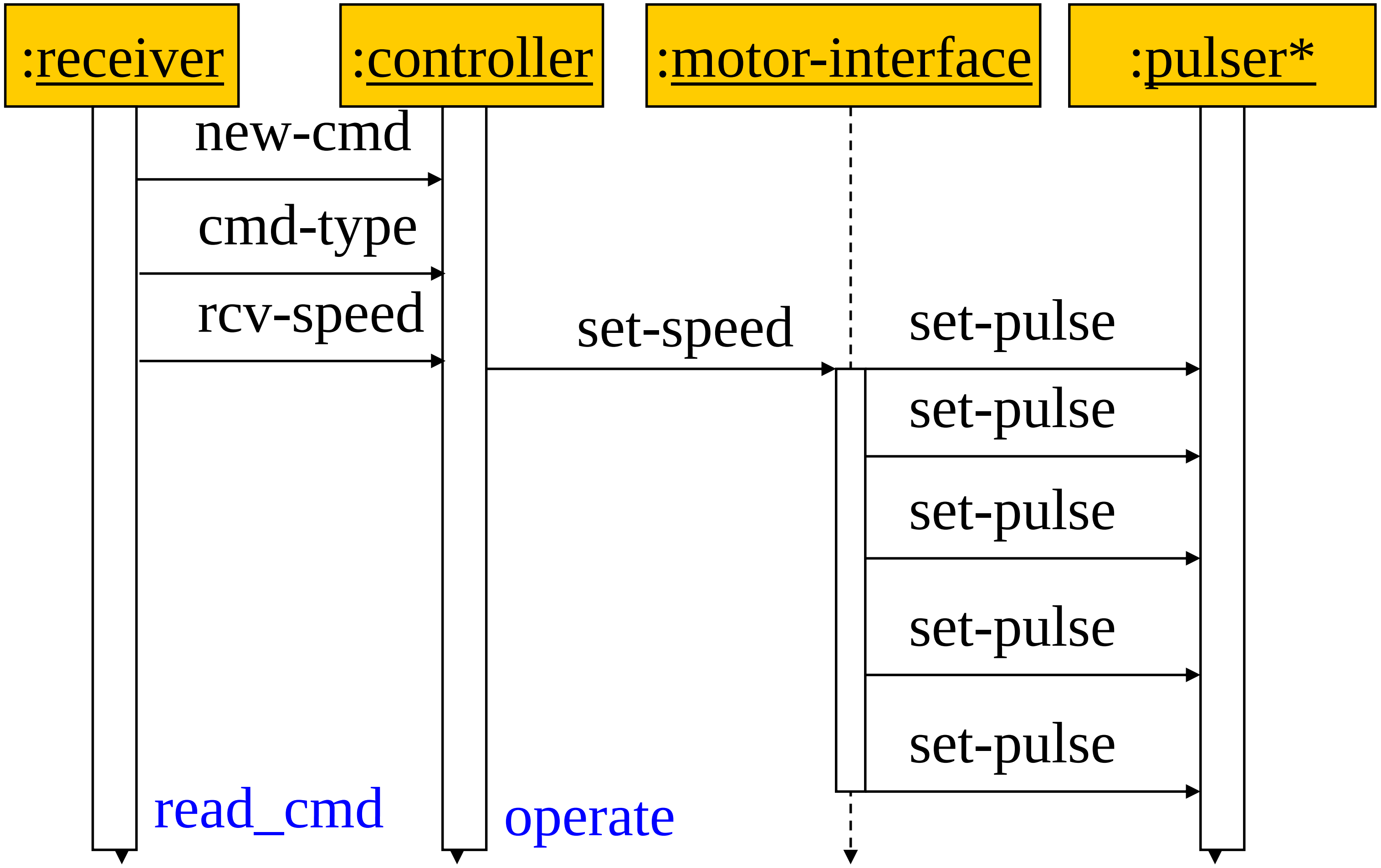
Controller class

controller
current-train: integer current-speed[ntrains]: integer current-direction[ntrains]: boolean current-inertia[ntrains]: unsigned-integer
operate() issue-command()

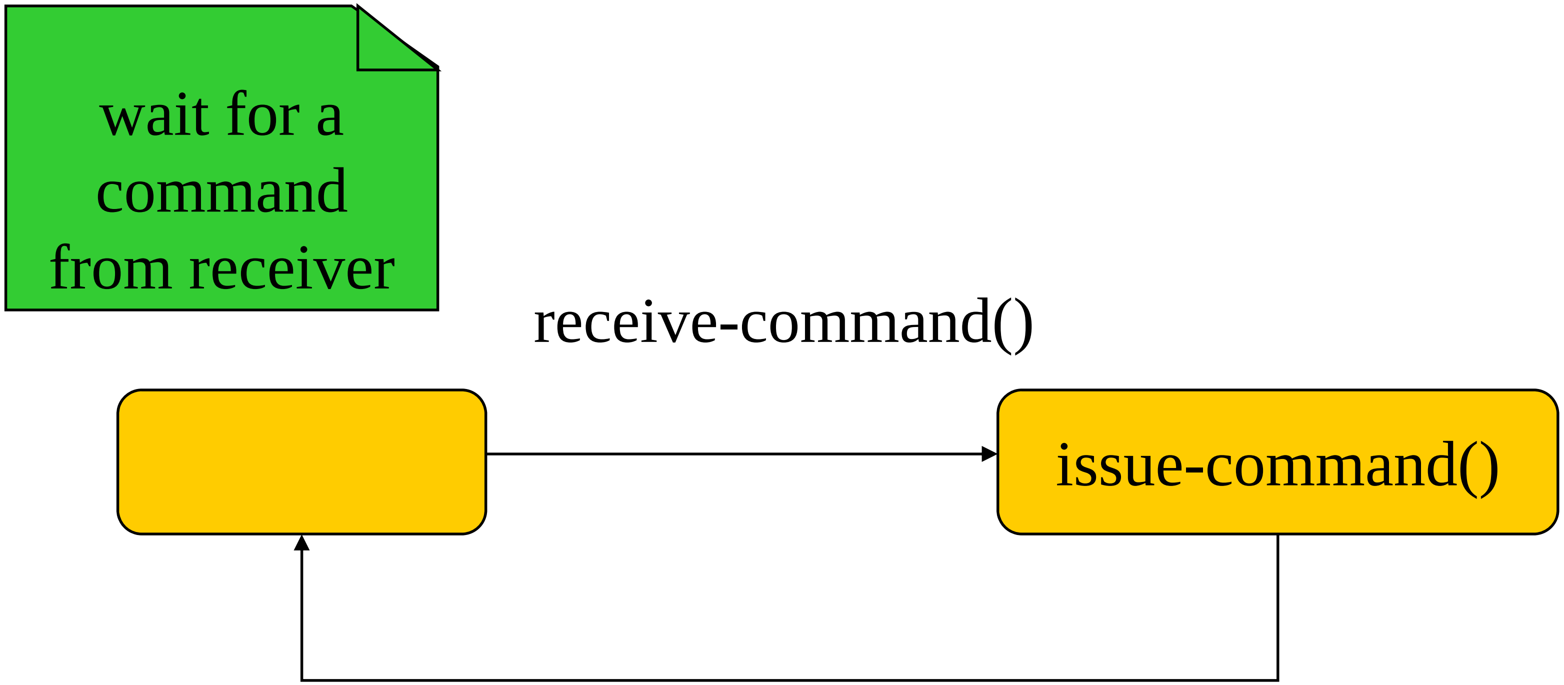
Setting the speed

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

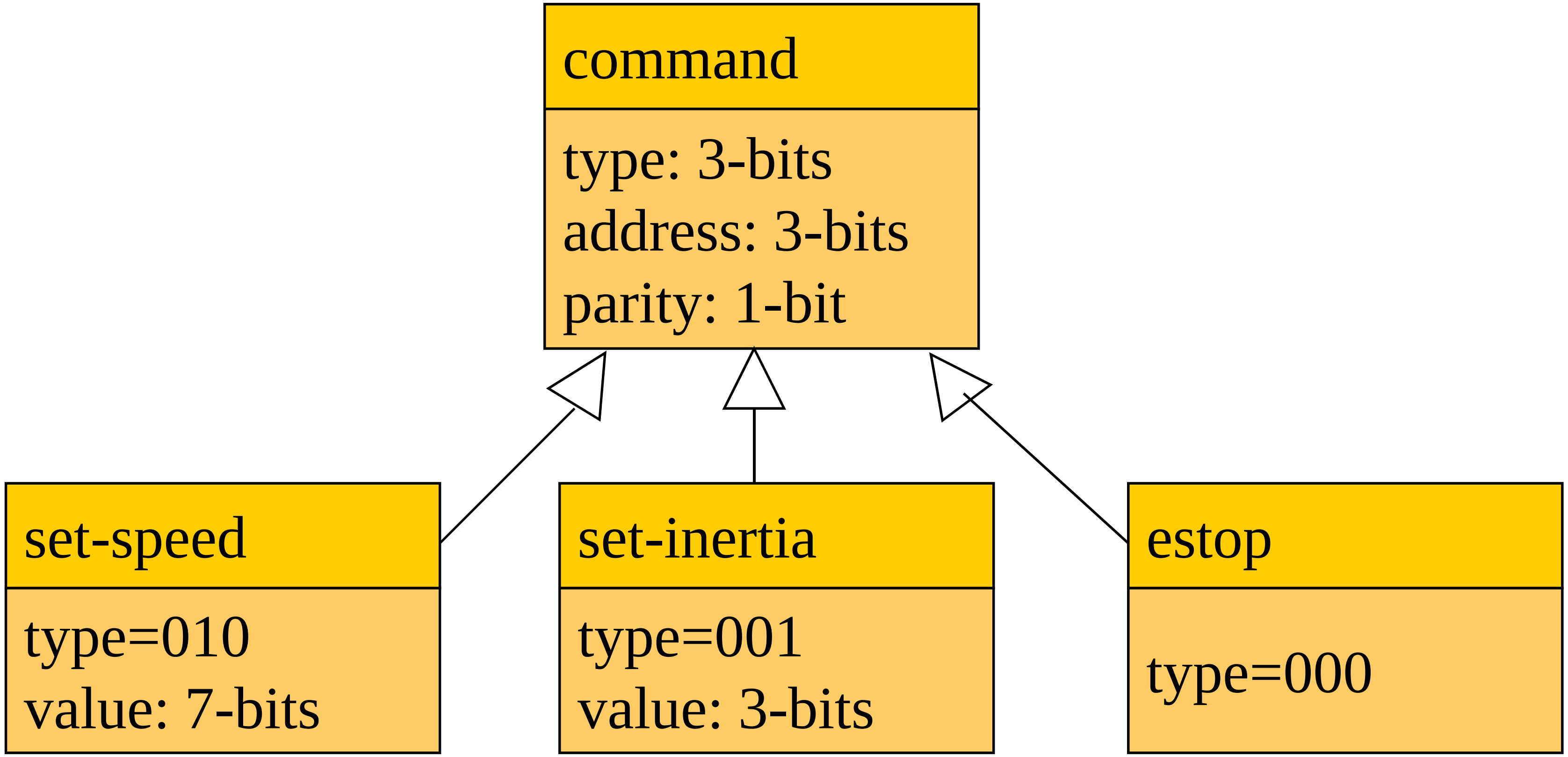
Sequence diagram for a set-speed command



Controller operate behavior



Refined command classes



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.