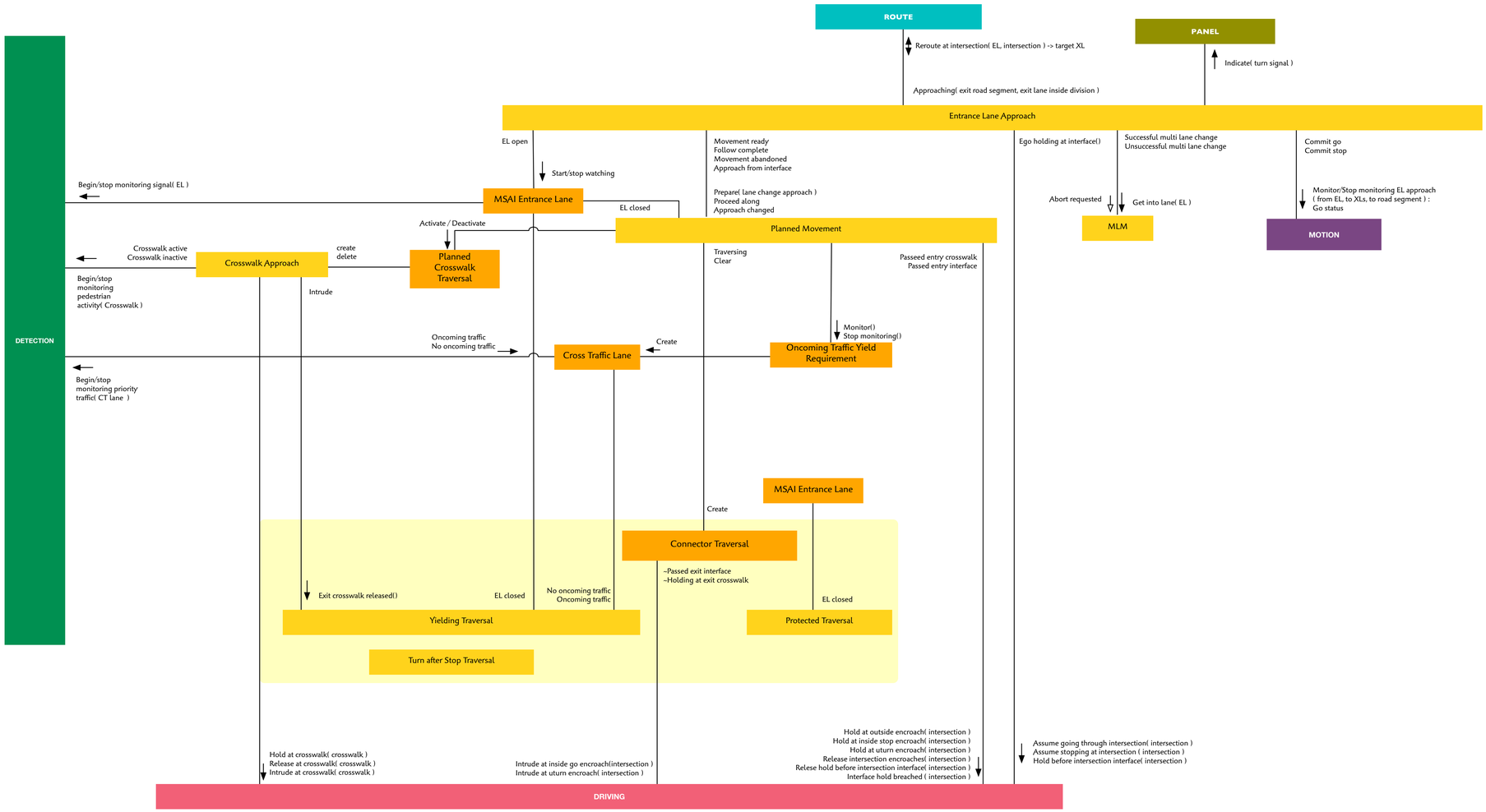
Signalized Intersection Traversal Class Collaboration Diagram

Parameter, data type and class name expansions:

EL = inside division, road segment
XL = inside division, road segment
exit conduit = conduit, road segment, intersection
target driving lane = inside division, outside division, road segment
interface = intersection, conduit, road segment
from EL = inside division, road segment
to XLs = { inside division }
CT Lane = inside division, road segment
turn signal : Turn Signal :: [left | right | cancel]
MSAI = Monitored Signalized Angled Intersection



Class Collaboration Diagram Key

This collaboration diagram primarily shows signal interactions among state machines, but also includes method invocation, domain and external entity operations and the setting of determinant attributes (boolean values that influence control).

lifecycle state machine defined on a class — Multi Lane Maneuver — yellow box

no state machine, but class has methods, determinant attributes or both — Oncoming Traffic Yield Requirement — orange box

external entity (a proxy for a domain external to our domain) — PANEL — colored box with all caps white text

A one or two way interaction path between the above elements

Each interaction is placed closest to its target on the interaction path

Any interaction that is not a signal generated by a state machine is associated with one of these symbols:

Asynchronous signal/event (no arrow as this is the default interaction, but a signal is placed adjacent to target class)

*Any signal prefaced by an asterisk *Get into driving lane(...), for example, is a creation signal triggering creation of the target instance*

Synchronous call (method or ext entity operation invocation) →

Call that returns a value ●→

Set/unset a determinant attribute →▷

Multiple classes belonging to the same generalization relationship are grouped by a light yellow rectangle.

A polymorphic event is signified with a ~. Such an event is addressed to the superclass, but handled in the subclasses.

Usually there is either a state machine for each subclass, with possible methods on the superclass as shown, OR just a state machine on the superclass and then methods possibly on the subclasses. It is usually bad form to have both state machines on super and subclasses, but sometimes necessary.