Ego Vehicle Subsystem Workbook

Vehicle Guidance Domain

Copyright © 2021 Toyota Research Institute, Inc.

Legal Notices and Information

Permission is hereby granted, free of charge, to any person obtaining a copy of these models and associated documentation files (the "Models"), to deal in the Models without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Models, and to permit persons to whom the Models is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Models.

THE MODELS ARE PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE Models OR THE USE OR OTHER DEALINGS IN THE MODELS.

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME
1.0	August 30, 2021	Initial external release of workbook.	LS and MAG

Contents

List of Figures

Chapter 1

Introduction

The Ego Subsystem models both the Ego Vehicle's general context (is it on the road or is it in a parking lot?) as well as the dynamic relationships necessary to characterize a specific maneuver.

General context

How do we establish the difference between an Ego Vehicle that is on the road vs. off-road? By formalizing relationships to certain elements defined in the Road Subsystem! For example, an Ego Vehicle on a Road (relationship R150) is, by definition, an On Road Ego Vehicle. If it is in a Driving Lane (relationship R151) it is defined as a Traveling Ego Vehicle with an Active Driving Lane.

High level maneuver

When the Ego Vehicle performs a high level maneuver, such as a lane change or an intersection approach (or even both simultaneously), it develops additional relationships with elements in the immediate driving environment. In the case of a lane change, these elements happen to be defined in the Road Subsystem. For example, our Traveling Ego Vehicle is assigned a target Driving Lane (relationship R157). This relationship is manifested as the Multi Lane Maneuver class. An instance of this class comes into existence and then executes each necessary Driving Lane Change until either the target is attained, in which case the target lane becomes the new Active Driving Lane, or it fails in one of numerous scenarios. In either case, the Multi Lane Maneuver instance terminates (deletes itself) upon completion.

Since the Ego Vehicle is typically in motion, it is continually developing relationships in this manner. Note however, that the avialable relationships depend entirely on the immediate driving environment.

Early release scope

In this early release of our driving environment models, the Multi Lane Change Maneuver is the only available behavior. But stay tuned, since signalized intersection traversal is coming up next!

Chapter 2

Class model

A complete Executable UML model consists of three interlocking facets. These are:

1. A class model which formalizes data/logic/constraints
2. A set of state models (synchronization)
3. A set of activities (computation)

In this chapter we focus on the class model facet. This facet defines the abstractions, data, logic, rules, policies and constraints that characterize a given subsystem.

The model consists of a diagram and a set of descriptions that document each element found on the diagram. A thumbnail of the diagram is included in this document for completeness, but we recommend getting your hands (or screen) on the full size version which should be easily located in the same repository.

Class descriptions appear first and are organized alphabetically for fast reference, like a dictionary. To make sense of these descriptions you probably want to start with a central class and then walk through the diagram looking up classes as you encounter them rather than reading from A to Z.

Each non-referential attribute (those without an R number) is described along with its class.

The next section describes all relationships in ascending numerical order. For each relationship there is a description of its meaning, multiplicity and how it is formalized in terms of referential attributes. Any additional constraints on the relationship are also included.

Class diagram

Here we focus on the Ego Vehicle itself and its relationship to the Road and Driving Lanes. We also specify the tunable parameters which direct how the Ego Vehicle behaves as it drives. For example, we can define the duration to indicate a turn before maneuvering into an adjacent Driving Lane.

In the current version we focus only on what we need to specify Multi Lane Maneuver.

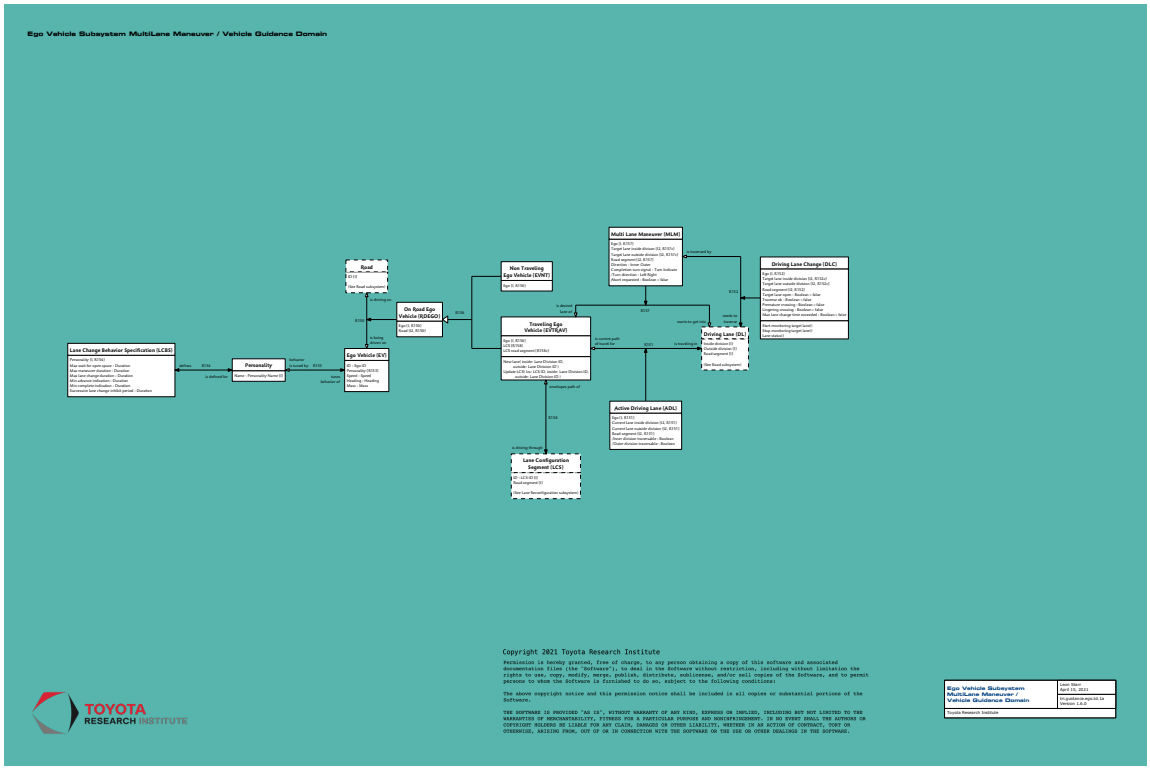

Figure 2.1: tri.guidance.ego.td.1a / Ego Vehicle Subsystem Class Diagram with a Multi-lane change focus

Please consult the associated Class Diagram while reading this document.

Classes

Active Driving Lane (ADL)

The Driving Lane currently and likely to remain occupied by a Traveling Ego Vehicle is considered the ‘active’ Driving Lane. During a lane change, the Ego Vehicle will have switched active lanes such that only one is active at a time even if both the source and target Driving Lanes are partially occupied. That’s where the ‘is likely to remain occupied’ characteristic comes in. There are many heuristics that can resolve this transition point. All of them, however, must ‘pick a lane’ such that we consider one the active lane at any given point in time.

Identifiers

1. Ego
2. Current lane inside division + Current lane outside division + Road segment

Since there can exist, at most one instance of Ego Vehicle, it stands to reason that either reference, given a value (or pair of values) will yield exactly zero or one instance. And it turns out that identifier #2 is particularly helpful for enforcing a key relationship constraint.

Attributes

Inner division traversable (derived)

Whether or not it is legally and physically possible to cross the Lane Division. This is based on the current Division Transition to the front inside of the Ego Vehicle. True if Soft Division and Soft Division.Passing ok is True, otherwise, false.

Type: Boolean

```
// Derived Attribute: Inner Division Traversable
// ---
// Class: Active Driving Lane
// ---
//
// Get the nearest soft division, if one exists, in front of the ego vehicle
// The nearest will be the one that is adjacent and extending out for some  ↩
    distance
//
soft .= /R151/R10/R19/R21/is the inside boundary of lane/Lane Division/
        Division Transition( -^Location )/Soft Division

not soft or soft and not Crossing ok?
    =>> False : =>> True
```

Outer division traversable (derived)

Same as the derivation for *Inner division traversable*.

Type: Boolean

```
// Derived Attribute: Outer Division Traversable
// ---
// Class: Active Driving Lane
// ---
//
// Get the nearest soft division, if one exists, in front of the ego vehicle
// The nearest will be the one that is adjacent and extending out for some  ↩
    distance
//
soft .= /R151/R10/R19/R21/is the outside boundary of lane/Lane Division/
        Division Transition( -^Location )/Soft Division

not soft or soft and not Crossing ok?
    =>> False : =>> True
```

Driving Lane Change

This is a behavior where the Ego Vehicle moves out of its Active (current) Driving Lane into an adjacent target Driving Lane. Ideally, the Ego Vehicle waits for space to open so that it can enter the target lane, indicates via turn signal in advance, enters the lane if it is still open (space available and traversal legal), continues to indicate and then waits a bit before any subsequent lane change.

A lot can go wrong with this ideal behavioral pattern. The driver may not signal adequately in advance, or a sudden intrusion into the target lane may force a return to the source lane. Nonetheless, we can anticipate and characterize both the ideal and non-ideal occurrences and stages in a Driving Lane Change.

Like all behaviors it terminates at some point successfully or unsuccessfully. An unsuccessful termination will be due to an undesired event or a timeout.

Identifiers

1. Ego
2. Target lane inside division + Target lane outside division + Road segment

Attributes

Target lane open

Whether or not there is space available in the target lane to permit entry. This does not take into account whether or not it is currently legal to traverse into the target lane. So, while the target lane may be clear of ado vehicles, it may yet be illegal or physically impossible to cross over.

Type: Boolean

Traverse ok

The target lane is clear of traffic and any dangerous obstacles, static or moving. This property applies to the target lane interior and not necessarily the interstitial Lane Division which may physically or legally forbid crossing.

(The external process that sets this value may consider the lane division, but we should not depend on that)

Type: Boolean

Premature crossing

The target lane was entered before adequate time had passed. We set this value true to record the suboptimal condition while carrying on with the maneuver.

Type: Boolean

Lingering crossing

The time taken to cross over the interstitial Lane Division was exceeded during traversal into the target lane.

Type: Boolean

Max lane change time exceeded

The overall time allotted for the Driving Lane Change maneuver was exceeded.

Type: Boolean

Ego Vehicle

There may be numerous other vehicles on the road, but we need to single out our driver's perspective. Our driver could be fully automated, semi-automated, or even a human. In all cases, the driving environment is defined relative to this point of view.

That's why we use the term Ego Vehicle rather than AV or 'automated vehicle' since the degree of automation is not our concern.

Identifiers

1. ID

It may seem odd to place an identifier on our one and only Ego Vehicle, but it ensures consistency within our relational modeling formalism. In other words, a set definition populated with a single element is algebraically no different than a set with zero or multiple instances. (Any generated code, on the other hand may make such a performance enhancing distinction if appropriate)

Attributes

ID

Nominal (always 1)

Speed

The current measured speed.

We assume that measured or otherwise detected or computed values are maintained by some service domain. We'll only use them when validating some environmental constraint such as whether or not the speed is above the current legal limit or if the mass is inconsistent with a permitted load on a bridge, etc.

Type: Speed (kph)

Heading

The current measured heading of the Ego Vehicle's forward (windshield) perspective.

Type: Heading (compass degrees)

Mass

The current assumed or measured mass. This definition will probably be updated and clarified later when we start modeling behavior involving it.

Type: Mass (Kg)

Lane Change Behavior Specification

This component of the Ego Personality defines configurable parameters relevant to making a Lane Change. At present, only Driving Lane Change behavior is being considered. Some of these parameters should be relevant to other types of lane change, however.

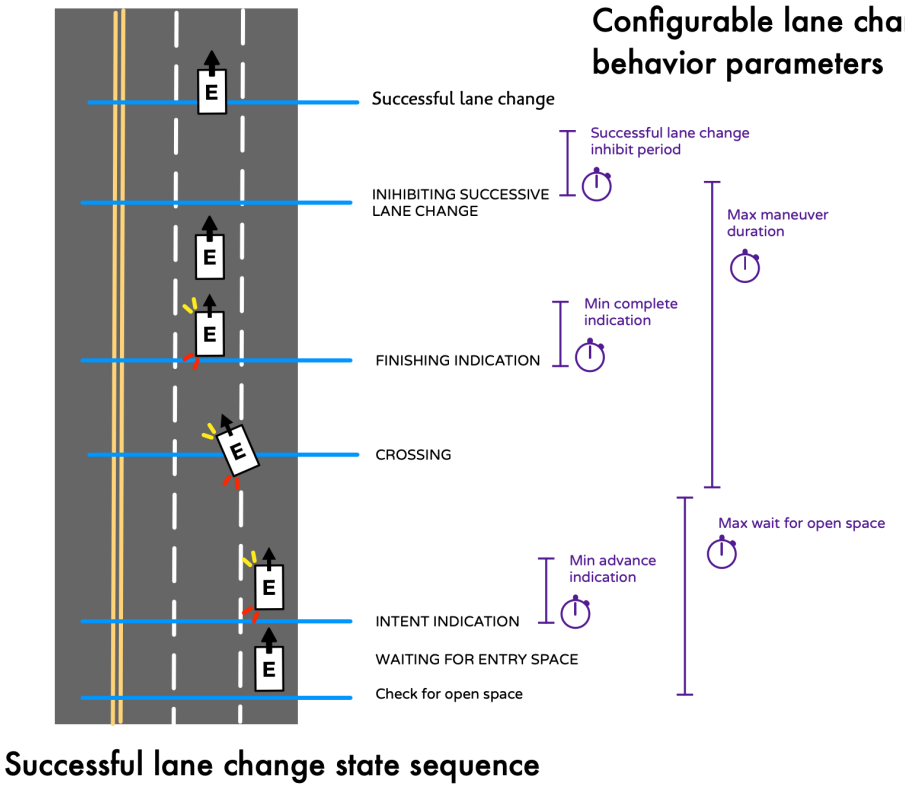

Figure 2.2: Lane Change Elements

Identifiers

1. Personality

Attributes

Max wait for open space

When attempting a Driving Lane Change or similar maneuver, this is the maximum amount of time to wait for a target Driving Lane to yield a feasible, safe opening in traffic where the Ego Vehicle may insert itself in that target lane. If this period expires, the Driving Lane Change will be aborted, and possibly attempted again at some future point in time.

Type: Duration

Max maneuver duration

During a Driving Lane Change, this is the maximum amount of time from the end of the advance indication interval (adequate advance turn signal indication) to the completion of the physical maneuver where the Ego Vehicle is aligned in the target Driving Lane.

Type: Duration

Max lane change duration

The maximum time it should take to get into the target lane once maneuvering begins from the source lane.

Type: Duration

Min advance indication

The minimum time interval where the turn signal is continuously on, indicating in the direction of the target Driving Lane in a Driving Lane Change, prior to any physical maneuver toward that target lane.

Type: Duration

Min complete indication

The minimum time interval where the turn signal remains on after crossing into the target lane before being turned off.

Type: Duration

Successive lane change inhibit period

After a successful Driving Lane Change, the Ego Vehicle must wait for at least this period of time before attempting another Driving Lane Change. The term “must” presumes that we maintain our definition of a Driving Lane Change. In the case of an assistive/supervisory autonomous product, where the human driver chooses to preempt the current Driving Lane Change either terminates with a logged error or assistive automation may forcefully delay the maneuver. In the case of either assitive or fully automated behavior, imminent danger of some type may abort the Driving Lane Change and trigger a different behavior, in which case the specified inhibit period is rendered moot. In other words, the “must” provision presumes that the Driving Lane Change behavior remains intact.

Type: Duration

Multi Lane Maneuver (MLM)

When a Traveling Ego Vehicle needs to get into a different Driving Lane, it begins this type of maneuver. In the simple case, the Ego Vehicle changes to a single adjacent Driving Lane. But any number of Lanes (including Non Driving Lanes) may be traversed in the process. It may be necessary, for example, to pass through a Dedicated Bike Lane on the way to the target Driving Lane.

At each point in this type of maneuver, there is always one Active Driving Lane representing the current travel path of the Traveling Ego Vehicle.

If successful, the maneuver will end with the Traveling Ego Vehicle in the requested target Driving Lane as its new Active Driving Lane. Otherwise, the maneuver aborts with the vehicle in the source or some intermediate Driving Lane as its Active Driving Lane.

Identifiers

1. Ego
2. Target lane inside division + Target lane outside division + Road Segment

Attributes

Direction

The direction from the Ego Vehicle toward the target Driving Lane. So if the Ego Vehicle is in the outermost lane, for example, and the target is the innermost lane, the direction would be inner.

Type: [inner | outer]

Completion turn signal

Upon successful completion of this maneuver, the turn signal will be left ‘successful’. If the maneuver is not successful, this value has no effect.

Type: Turn Indicate

/Turn direction

Conversion of the Direction attribute value to left or right taking into account the Ego’s Traffic Territory (USA, Japan, etc)

Type: [left | right]

Abort requested

If the requester, such as Entrance Lane Approach, wishes to terminate the Multi Lane Maneuver, it sets this value to `true`. This is necessary since an individual Lane Transition will need to complete before the termination request can be processed.

Type: Boolean

Non Traveling Ego Vehicle

An On Road Ego Vehicle that is not currently in any Driving Lane or transitioning in or out of a Driving Lane is, by our definition, not traveling. A Non Traveling Ego Vehicle may be coasting along in a Shoulder Lane or parking in a Parking Lane, for example.

Identifiers

1. Ego

Attributes

(No non-referential attributes)

On Road Ego Vehicle

Here we distinguish an Ego Vehicle driving on a Road as distinct from an off road Ego Vehicle which might be in a parking garage or driveway.

While it is certainly true that many vehicle behaviors apply equally well on the road or in a parking lot, there are many which vary. For example, you cannot perform a lane change in a parking lot. There are no shared bike lanes, shoulders, or parking lanes (as they are defined in the Road Subsystem anyway). You do not encounter a signalized intersection in a parking lot. But wait, could you? It certainly is not difficult to imagine such an arrangement, but it would be a rare exception. And, in such an exception, we might find a way to define some kind of road equivalent internal to such an elaborate parking lot.

Until we model parking the parking lot environment and clarify what's in and out, it is safest to assume that these are completely different environments, each with their own specific rules and geometries. By singling out an Ego Vehicle on Road, we emphasize that this is the only environment that we are defining and that, outside this environment, our driving environment assumptions do not necessarily apply.

Identifiers

1. Ego
2. Road

Since there can exist at most one instance of Ego Vehicle on Road, it stands to reason that either attribute, given a value, will yield exactly zero or one instance.

Attributes

(No non-referential attributes)

Personality

There are a variety of tunable parameters that influence how the Ego Vehicle will behave during certain maneuvers or and conditions. It is helpful to specify multiple sets of such parameters so that we can test.

Identifiers

1. Name

Attributes

Name

A name that reflects the purpose of a given parameter set such as "Conservative", "Polite Canadian", "Urban aggressive", "Passive Aggressive Swedish", "Press Demo", etc.

Type: Personality Name based on system Name type

Traveling Ego Vehicle

When an Ego Vehicle is in a Driving Lane (or transitioning between them) it is said to be ‘traveling’. Whether or not the Ego Vehicle is actually moving is immaterial. It could be stopped in traffic or waiting for a light to change, for example.

Identifiers

1. Ego

Attributes

(No non referential attributes)

Relationships

R150

- **Ego Vehicle** is driving on *zero or one* **Road**
- **Road** is being driven by *zero or one* **Ego Vehicle**

At a given point in time, the Ego Vehicle may or may not be on the Road. Our vehicle could be in a parking lot, in a driveway or in the middle of a grassy field. Numerous constraints apply to an Ego Vehicle only when it is, in fact, on a Road somewhere, so we need to recognize and carefully distinguish when this is in fact the case. This frees us up to make different rules to constrain vehicle behavior when the Ego Vehicle is in a non-road environment.

An Ego Vehicle cannot simultaneously drive on multiple Roads. This is largely due to the way that Road is defined. For example, a Road may have multiple map designations. So the fact that we are on "Highway 1", which is also "CA-1" which is also the "Pacific Coast Highway" constitutes a single Road with multiple designations and not three distinct Roads. What about turning through an Intersection? At some, possibly arbitrary, point in the turn we must decide that the Ego Vehicle has left one Road and entered the other. But at no point in time will we consider the Ego Vehicle to be on both Roads simultaneously. (Note that not all intersection turns involve leaving the current Road!)

A Road may be detected by the Ego Vehicle even though the Ego Vehicle is not driving on it: an intersecting Road up ahead, for example.

It is true that a Road may be simultaneously driven by multiple automated vehicles, but the term "Ego" is critical. By definition, there is only one Ego Vehicle, ourself, and therefore we either are or are not driving on a given Road. Thus, a Road is being driven on at most one Ego Vehicle.

Formalization

On Road Ego Vehicle.Ego →Ego Vehicle.ID On Road Ego Vehicle.Road → Road.ID

R151

- **Traveling Ego Vehicle on Road** is traveling in *one* **Driving Lane**
- **Driving Lane** is current path of travel for *zero or one* **Traveling Ego Vehicle**

An Ego Vehicle On Road is certainly on the Road, but could be in a Non Driving Lane. The vehicle may be pulled over on the shoulder or it may be in a Shared Single Road Lane. In the latter case, a lane change would be out of the question since there is only one lane in the Road. So, the fact that a Traveling Ego Vehicle is in a Driving Lane is a key qualification for a number of possible behaviors.

A given Driving Lane may or may not be currently occupied by the Ego Vehicle at the moment.

Formalization

Active Driving Lane.Ego → Traveling Ego Vehicle.Ego Active Driving Lane.(Current lane inside division, Current lane outside division, Road segment) → Driving Lane.(Inside division, Outside division, Road segment)

R152

- **Multi Lane Maneuver** needs to traverse *one* **Driving Lane**
- **Driving Lane** is traversed by *zero or one* **Multi Lane Maneuver**

Once initiated, a Multi Lane Maneuver targets an initial Driving Lane destination. This process keeps repeating until the final target is attained, at which point the maneuver no longer exists. So throughout a Multi Lane Maneuver's existence, there is always a target lane to get into (traverse).

At any point in time, a given Driving Lane may or may not be the target of a Multi Lane.

Formalization

Driving Lane Change.Ego → Multi Lane Maneuver.Ego

Driving Lane Change.(Target lane inside division, Target lane outside division, Road segment) → Driving Lane.(Inside division, Outside division, Road segment)

Constraints

The Driving Lane.Target lane inside division value matches a /R152/R151/OR34/Driving Lane Order.Inside division value one rank in either ascending or descending order.

Target lane constraint

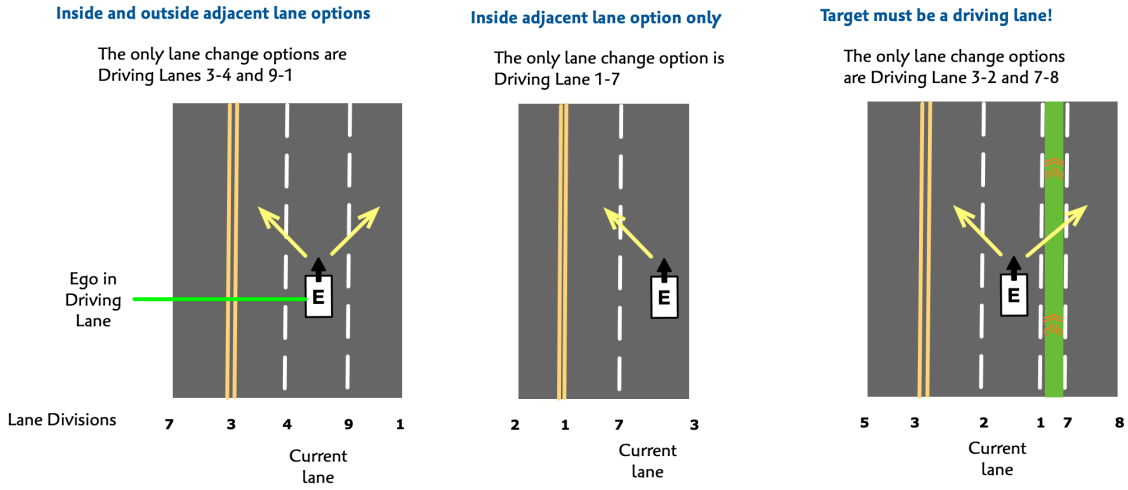

Figure 2.3: Target Driving Lane constraint

The Driving Lane.Target outside division value matches a /R152/R151/OR34/Driving Lane Order.Outside division value one rank in either ascending or descending order.

R153

- **Ego Personality** tunes behavior of *zero or one* **Ego Vehicle**
- **Ego Vehicle** behavior is tuned by *one* **Personality**

The Ego Personality represents the collection of parameters that configure the behavior of the Ego Vehicle. We may define multiple personalities for diagnostic purposes. Or, we may need to deploy our automated vehicles with a choice of personalities that can be selected in the field. The important point, though, is that, at any given time, the Ego Vehicle as exactly one controlling personality tuning its behavior.

Assuming multiple personalities have been defined, from the perspective of each, it either is or is not the controlling personality.

Formalization

Ego Vehicle.Personality → Personality.Name

R154

- **Personality** defines *one* **Lane Change Behavior Specification**

- **Lane Change Behavior Specification** is defined for *one* **Personality**

There will be many components to the Personality. For now, there is only one and that is the Lane Change Behavior Specification. The specification cannot stand alone as it is part of the overall personality package.

Formalization

Lane Change Behavior Specification.Personality → Personality.Name

R156

- **On Road Ego Vehicle** is a **Traveling or Non Traveling Ego Vehicle**

When an On Road Ego Vehicle is in a Driving Lane it is considered to be traveling. Otherwise it is not.

Certain driving rules and policies apply when an On Road Ego Vehicle is traveling, such as the requirement that it always be in a Driving Lane (or transitioning between them) that don't apply to a vehicle that may be parked or coasting in a Road Segment somewhere like a Shoulder or Exclusion Lane. In particular, we can always ask an On Road Ego Vehicle, 'What Driving Lane are you in?' and always get an answer, even during a lane transition.

Formalization

<subclass>.Ego → On Road Ego Vehicle.Ego

R157

- **Traveling Ego Vehicle** wants to get into *zero or one* **Driving Lane**
- **Driving Lane** is desired lane of *zero or one* **Traveling Ego Vehicle**

Only one Multi Lane Maneuver may be executed at a time by a Traveling Ego Vehicle. So either the vehicle wants to traverse toward some target Driving Lane or it doesn't.

Any given Driving Lane may or may not be the ultimate target of a Multi Lane Change maneuver at a given time.

Notice that a Multi Lane Maneuver can only exist if an ultimate destination Driving Lane has been selected.

Formalization

Multi Lane Maneuver.Ego → Traveling Ego Vehicle.Ego Multi Lane Maneuver.(Target lane inside division, Target lane outside division, Road segment) → Driving Lane.(Inside division, Outside division, Road segment)

R158

- **Traveling Ego Vehicle** is driving through *one* **Lane Configuration Segment**
- **Lane Configuration Segment** envelopes path of *zero or one* **Traveling Ego Vehicle**

Imagine a Traveling Ego Vehicle at some point within a Road Segment. Let's say that it is not against the flow of traffic illegally, so it must be in the forward Conduit. Let's say it is a multi-lane Conduit, so the vehicle sees the Driving Lane it is in along with possible Lanes to its left and/or right within its Conduit.

Since the Lane Configuration Segment defines an unchanging configuration of Lanes in a Conduit, and a Road Segment is fully partitioned as a set of adjacent Lane Configuration Segments, we know that our Traveling Ego Vehicle must be in some specific Lane Configuration Segment.

Ah, but what if a right turn lane is beginning to open up just to the right of our Traveling Ego Vehicle? Could we say that we are crossing over from one Lane Configuration Segment to the next and therefore temporarily spanning two?

We could, but we don't. By policy we will always consider the Traveling Ego Vehicle to be in one of them. As soon as the front of the vehicle (or some distance forward of the vehicle) crosses into the next Lane Configuration Segment, it becomes the current Lane Configuration Segment.

At any point in time a given Lane Configuration Segment may or may not be occupied by the Traveling Ego Vehicle.

Formalization

Traveling Ego Vehicle.(LCS, LCS road segment) $\rightarrow$ Lane Configuration Segment.(ID, Road segment)

Chapter 3

Behavior model

The second facet of a complete Executable UML model is a set of state models.

Lifecycle state models

Each state model formalizes the lifecycle of a class. Only those classes whose instances evolve through a dynamic lifecycle require a state model. The lifecycle of a class whose instances are simply created and deleted or just persist, do not benefit from a state model. In fact, some subsystems may not require any state models at all. In such a subsystem attributes are updated, instances are created, accessed and deleted and not much else happens.

State machine diagram

A state model is best conceived as a diagram of states interconnected by transitions. It serves as a nice visual representation of the instance lifecycle. The state machine diagram is incomplete, however, since it does not show what happens when an event that does not trigger a transition occurs in a given state.

State table

A state table, on the other hand shows the transition, ignore or error response for all events in all states defined for a class. Since the state table is complete, only it is used when generating code. Nonetheless, the diagram is retained and kept in sync with the table since it is the easiest way to evaluate and understand the scope and purpose of instance behavior.

Driving Lane Change state model

The following diagram shows the state model for the **Driving Lane Change** class.

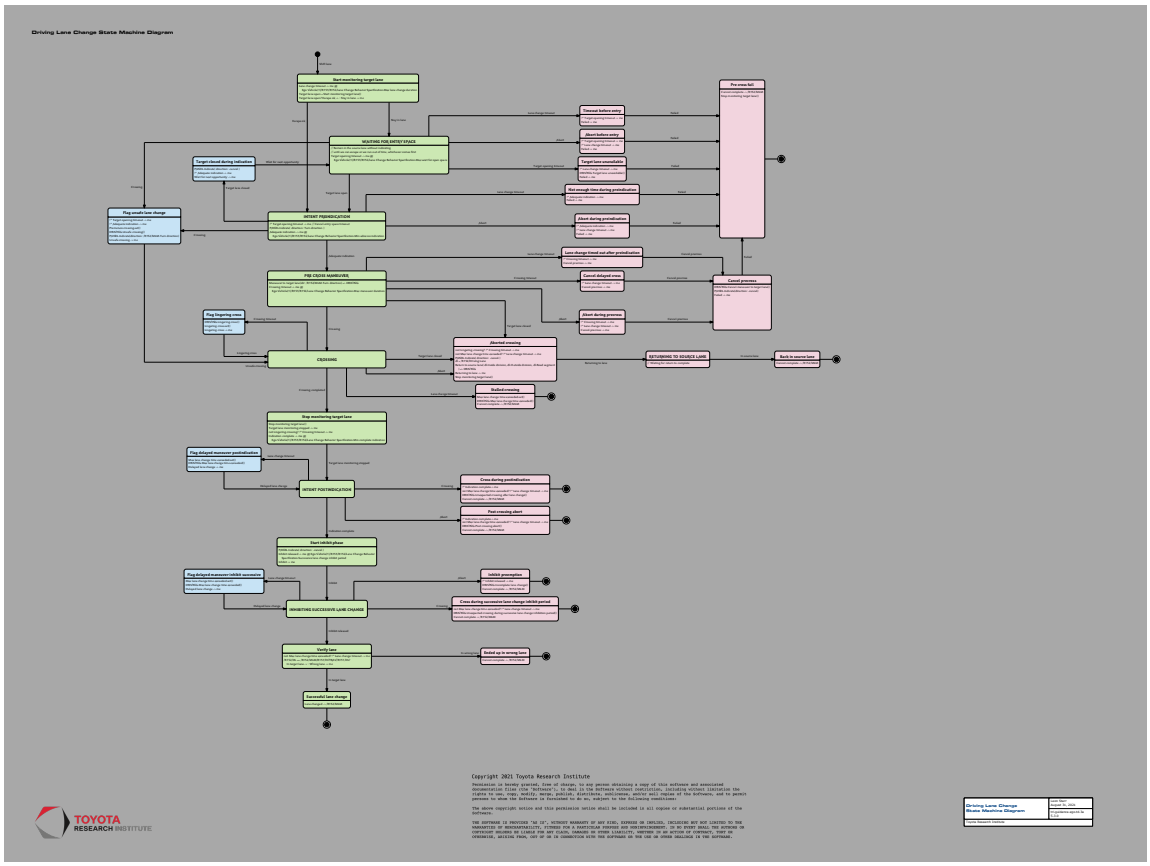

Figure 3.1: Driving Lane Change State Machine Diagram

Driving Lane Change state transitions

The following table shows the transition matrix for the **Driving Lane Change** class.

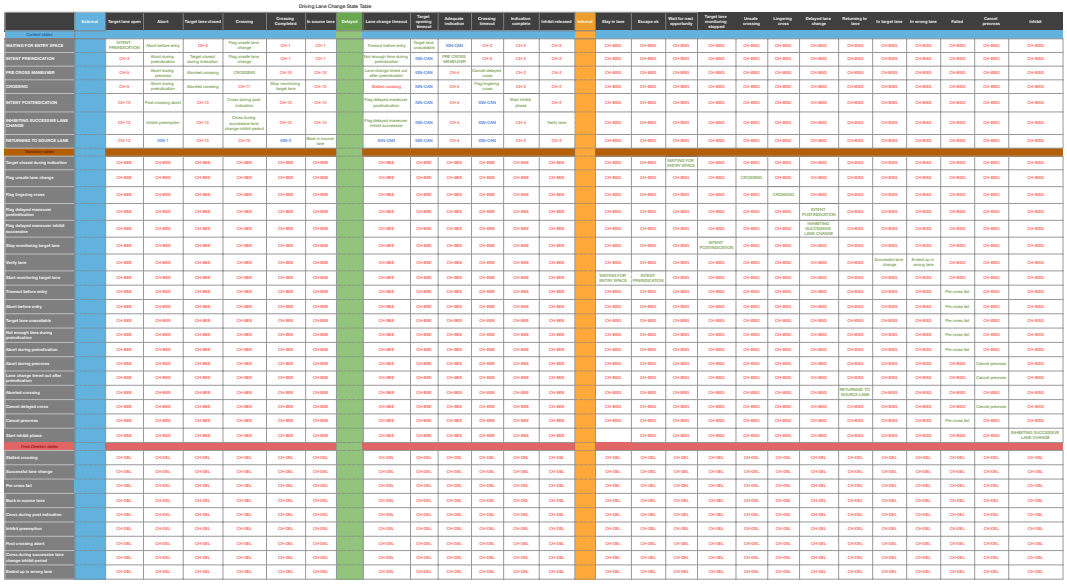

Figure 3.2: Driving Lane Change State Transition Table

Comments on non-transition event responses in the state transition table

Each cell in the table represents an instance's response to an event as a transition (green), ignore (blue) or can't happen (red).

A transition response will appear on the state machine diagram.

An ignore response indicates that if the event occurs in the given state it will be ignored. This is not an error, but a normal behavior. The explanation of why this is a valid response is tagged in the cell as IGN<number> and explained in the associated table of non-transition response comments below.

A can't happen response indicates that our model is broken if that event actually does occur. When 'can't happens' occur in running code, we need to go back and debug our models. The explanation of why this is a valid response is tagged in the cell as CH<number> and explained in the associated table of non-transition response comments below.

Comment	Description
IGN-1	As an abort is effectively in process we can safely ignore this event.

Comment	Description
IGN-2	This is a lingering event that may have happened at the same time as an abort or target lane closed. We can safely discard it since we are waiting to find out if we are in the source lane.
IGN-3	We're already in the target lane, so it doesn't matter now.
IGN-4	We've already aborted, so there's nothing else to do.
IGN-5	Doesn't matter since we've aborted the lane change.
IGN-6	This could be the crossing back over toward the target lane, so we just wait until fully in the source lane.
IGN-7	At this point, we are committed unless we get an explicit return to source lane command.
IGN-CAN	We can safely ignore a delayed event arriving after it has been canceled.
CH-1	We must be in the source lane upon entry to this state. If we get this event before a Crossing event, then something is wrong.
CH-2	Delayed event could not have been requested before entering this state.
CH-3	Coordination with the external source should be configured to ensure that we can't get two consecutive events of this particular event. So, if we got an open here, it means that it wasn't previously which is really bad, since we shouldn't have gotten into this state in the first place. (Make sure the domain operations/class methods are properly specified/translated).
CH-4	This delayed event was canceled or expired before entering this state.
CH-5	It had to be open to get us into this state, and if it closes, we leave the state. A second open may indicate a problem in the interface domain operations or a class method.
CH-6	Similar to CH-5, but with a different event. We're crossing which means that we were already in the source lane. Next we should get an in target lane which takes us to a new state before another in source lane occurs.
CH-7	We can't be in the source lane at this point. Of course, this lane may become the source lane in a future lane change, but not this one.
CH-8	We already got that event (duplicates should be screened out).
CH-9	It is not possible to abort if we've made it all the way into the target lane.
CH-10	Causality. A crossing event must always preceed an *in the source lane* event.
CH-11	We already got the crossing event. It is an error if the source of this event sends more than one prior to initiating another lane change or aborting somehow.
CH-12	We should end up in the target lane unless we are forced to return in which case we would have gotten an abort or return to source lane first.
CH-13	Target lane monitoring has been stopped, so this event cannot arrive in this state.
CH-14	We can only end up in the source lane via an aborted crossing.
CH-15	If we are in this state, we know that crossing was never completed. So, at worst, the next event should be a lingering *Crossing completed.*
CH-BSG	Can't happen (Blind to Self Generated event from other state) Self generated event in other state cannot be seen here.
CH-DEL	Can't happen, the instance is deleted in this state as soon as the state's activity has completed.
CH-BEE	Blind to External Events. This transient state reacts to its own self generated event only. So it should never see a non-self generated event.

States

The third facet of a complete Executable UML model is a set of activities. An activity may perform a computation, access attributes, manage temporary variables, communicate with external entities and send immediate or delayed signals to other state machine instances. Every state defines a state activity which consists of a set of zero or more actions. Therefore, some states have no actions at all and simply represent a context.

Context states

States named with all uppercase letters represent a period of time where an instance waits for some delayed (time based) or external event to arrive. Such a state represents the current context of the instance.

CROSSING

What's going on in this state

Some part of the Ego Vehicle has breached the adjacent Lane Division in the direction of the target lane.

Informal action taken

No actions. We are just waiting for the Ego Vehicle to fully enter the target lane.

INHIBITING SUCCESSSIVE LANE CHANGE

What's going on in this state

We cancel the turn signals and, for safety, we wait a bit before executing any additional lane changes.

Informal action taken

No actions.

INTENT POSTINDICATION

What's going on in this state

We're in the target Driving Lane with turn signals still on for a bit.

Informal action taken

No actions.

INTENT PREINDICATION

What's going on in this state

We are signaling our intention to get into the target Driving Lane.

Informal action taken

Cancel the opening timeout delayed event to self and tell PANEL to signal in our turn direction. Send an event to self delayed by the minumum advance indication in our active lane change spec.

Action

```
!* Target opening timeout -> me // Cancel entry space timeout PANEL.Indicate ↩
    ( direction: Turn direction )
Adequate indication -> me @
    Ego Vehicle(1)/R153/R154/Lane Change Behavior Specification.Min advance  ↩
        indication
```

PRE-CROSS MANEUVER

What's going on in this state

The turn signal is still on and we are moving toward the interstitial Lane Division with the intention to cross into the Target Lane.

Informal action taken

We confirm with DRIVING that we are ready to cross in our turn direction and send an event to self delayed by the maximum maneuver duration in our active lane change spec.

Action

```
Maneuver to target lane(dir: /R152/MLM.Turn direction) => DRIVING
Crossing timeout -> me @
    Ego Vehicle(1)/R153/R154/Lane Change Behavior Specification.Max maneuver ↩
         duration
```

RETURNING TO SOURCE LANE

What's going on in this state

We are maneuvering back to our original Driving Lane after a failed crossing.

Informal action taken

No actions, just waiting to get fully back in our original lane

WAITING FOR ENTRY SPACE

What's going on in this state

We are in the source Driving Lane waiting for it to be okay to enter the target Driving Lane

Informal action taken

Send an event to self delayed by the target opening timeout in the active lane change behavior specification.

Action

```
// Remain in the source lane without indicating
// until we can escape or we run out of time, whichever comes first

Target opening timeout -> me @
    Ego Vehicle(1)/R153/R154/Lane Change Behavior Specification.Max wait for ↩
        open space
```

Transitory States

A state is transitory if it is exited as soon as its activity completes on some event that the instance sends to itself. In Executable UML any self directed event is processed ahead of any pending external event. This makes it possible, for example, to perform an if-then evaluation and then exit on either condition with the appropriate self directed event without the possibility of external interference.

Abort before entry

Action

```
!* Target opening timeout -> me
!* Lane change timeout -> me
Failed -> me
```

Aborted Crossing

What's going on in this state

Crossing cannot be completed

Informal action taken

Cancel the lingering cross delayed event to self if not already fired. Cancel the max lane change time exceeded delayed event to self if not already fired. Cancel the turn signal. Notify DRIVING that we are returing to the source lane. Stop monitoring for an opening in the target lane

Action

```
not Lingering crossing? !* Crossing timeout -> me
not Max lane change time exceeded? !* Lane change timeout -> me
PANEL.Indicate( direction: .cancel )
dl = /R152/Driving Lane
Return to source lane( dl.Inside division, dl.Outside division, dl.Road  ↩
    segment
    ) => DRIVING
Returning to lane -> me
Stop monitoring target lane()
```

Abort During Pre-Cross

Action

```
!* Crossing timeout -> me
!* Lane change timeout -> me
Cancel precross -> me
```

Abort During Pre-Indication

Action

```
!* Adequate indication -> me
!* Lane change timeout -> me
Failed -> me
```

Cancel Delayed Cross

Action

```
!* Lane change timeout -> me
Cancel precross -> me
```

Cancel Pre-Cross

Action

```
DRIVING.Cancel maneuver to target lane()
PANEL.Indicate(direction: .cancel)
Failed -> me
```

Flag Delayed Maneuver Inhibit Successive

What's going on in this state

The lane change maneuver timed out while inhibiting the next maneuver.

Informal action taken

Notify DRIVING, set the appropriate boolean attribute and remain in the INHIBIT SUCCESSIVE LANE CHANGE state

Action

```
Max lane change time exceeded.set()
DRIVING.Max lane change time exceeded()
Delayed lane change -> me
```

Flag Delayed Maneuver Post-Indication

What's going on in this state

The entire lane change maneuver period has been exceeded for some reason while in the INTENT POSTINDICATION state.

Informal action taken

Notify DRIVING, set the appropriate boolean attribute and remain in the INTENT POSTINDICATION state

Action

```
Max lane change time exceeded.set()
DRIVING.Max lane change time exceeded()
Delayed lane change -> me
```

Flag Lingering Cross

What's going on in this state

It was taking too long to cross over the interstitial Lane Division for some reason.

Informal action taken

Set the appropriate boolean attribute to record this fact and remain in CROSSING state.

Action

```
DRIVING.Lingering cross()
Lingering cross.set()
Lingering cross -> me
```

Flag Unsafe Lane Change

What's going on in this state

A crossing event has occurred before an adequate pre-indication period.

Informal action taken

Cancel the target opening and adequate indication delayed events to self. Note that a premature crossing has occurred by setting the appropriate boolean attribute. Notify driving and activate the turn signals in the turn direction. Proceed to the CROSSING state

Action

```
!* Target opening timeout -> me
!* Adequate indication -> me
Premature crossing.set()
DRIVING.Unsafe crossing()
PANEL.Indicate(direction: /R152/MLM.Turn direction)
Unsafe crossing -> me
```

Lane Change Timed Out After Pre-Indication

Action

```
!* Crossing timeout -> me
Cancel precross -> me
```

Not Enough Time During Pre-Indication

Action

```
!* Adequate indication -> me
Failed -> me
```

Start Inhibit Phase

What's going on in this state

Prep transition into INHIBIT phase

Informal action taken

Cancel the turn signal and send an event to self delayed by the inhibit period found in the active lane change spec

Action

```
PANEL.Indicate( direction: .cancel )
Inhibit released -> me @ Ego Vehicle(1)/R153/R154/Lane Change Behavior
    Specification.Successive lane change inhibit period
Inhibit -> me
```

Start Monitoring Target Lane

What's going on in this state

We initialize the newly created Driving Lane Change instance

Informal action taken

Send an event to self delayed by the total time permited for a single Driving Lane Change as defined in the active lane change spec. Begin monitoring the target lane and transition to next state based on an immediate evaluation.

Action

```
Lane change timeout -> me @
    Ego Vehicle(1)/R153/R154/Lane Change Behavior Specification.Max lane  ↩
        change duration

Target lane open = Start monitoring target lane()
Target lane open? Escape ok -> : Stay in lane -> me
```

Stop Monitoring Target Lane

What's going on in this state

Now that the crossing is complete there is no need to monitor the target lane for an opening anymore

Informal action taken

Cancel monitoring by triggering the local method. Cancel the Crossing timeout event to self if still pending. Send an event to self delayed by the mimumum completion signaling interval as determined by our active lane change specification

Action

```
Stop monitoring target lane()
Target lane monitoring stopped -> me
not Lingering crossing? !* Crossing timeout -> me
Indication complete -> me @
    Ego Vehicle(1)/R153/R154/Lane Change Behavior Specification.Min complete ↩
        indication
```

Target Closed During Indication

What's going on in this state

While signaling, but before maneuvering the target lane became unavailable. We cancel our turn signal and wait for another opportunity

Informal action taken

Cancel turn signal and delayed Adequate indication event

Action

```
PANEL.Indicate( direction: .cancel )
!* Adequate indication -> me
Wait for next opportunity -> me
```

Target Lane Unavailable

Action

```
!* Lane change timeout -> me
DRIVING.Target lane unavailable()
Failed -> me
```

Timeout Before Entry

Action

```
!* Target opening timeout -> me
Failed -> me
```

Verify Lane

What's going on in this state

Did we end up in the correct target lane?

Informal action taken

We're done, so cancel the max maneuver delayed event to self, if it hasn't fired already. Also check to see if the intended target lane is our active driving lane. If not, fail the maneuver.

Action

```
not Max lane change time exceeded? !* Lane change timeout -> me
/R152/DL == /R152/MLM/R157/EVTRAV/R151/DL?
    In target lane -> : Wrong lane -> me
```

Final deletion states

In final deletion states, the class instance are automatically deleted after the activity is performed.

Back in source lane

Action

```
Cannot complete -> /R152/MLM
```

Cross During Post-Indication

Action

```
!* Indication complete-> me
not Max lane change time exceeded? !* Lane change timeout -> me
DRIVING.Unexpected crossing after lane change()
Cannot complete -> /R152/MLM
```

Cross During Successive Lane Change Inhibit Period

Action

```
not Max lane change time exceeded? !* Lane change timeout -> me
DRIVING.Unexpected crossing during successive lane change inhibition period ↩
    ()
Cannot complete -> /R152/MLM
```

Ended Up in Wrong Lane

Action

```
Cannot complete -> /R152/MLM
```

Inhibit Preemption

Action

```
!* Inhibit released -> me
DRIVING.Incomplete lane change()
Cannot complete -> /R152/MLM
```

Post-Crossing Abort

Action

```
!* Indication complete-> me
not Max lane change time exceeded? !* Lane change timeout -> me
DRIVING.Post crossing abort()
```

Pre-Cross Fail

Action

```
Cannot complete -> /R152/MLM
Stop monitoring target lane()
```

Successful lane change

What's going on in this state

We have successfully completed the Driving Lane Change

Informal action taken

Notify the MultiLane Maneuver and then delete self

Action

```
Lane changed -> /R152/MLM
```

Stalled crossing

Action

```
Max lane change time exceeded.set()
DRIVING.Max lane change time exceeded()
Cannot complete -> /R152/MLM
```

Multi Lane Maneuver state model

The following figure shows the state model diagram for the **Multi Lane Maneuver** class.

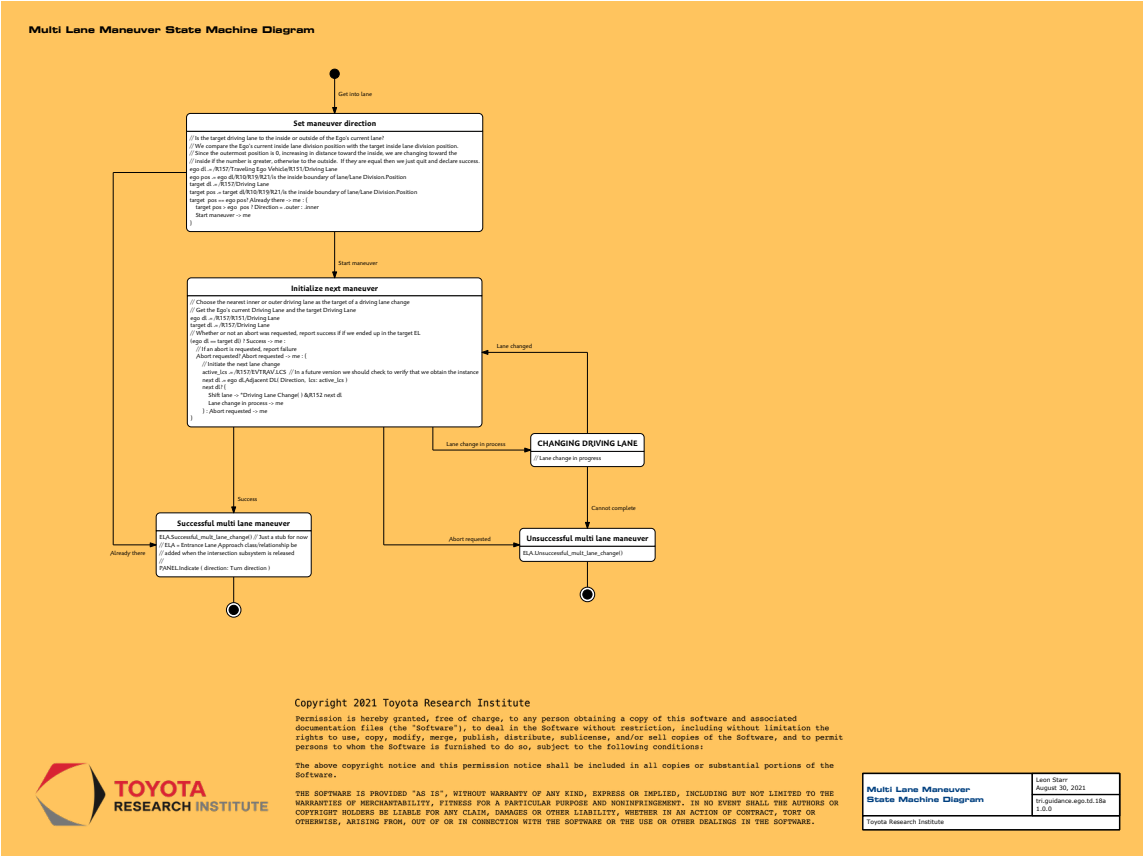

Figure 3.3: Multi Lane Maneuver State Machine

Multi Lane Maneuver state transitions

The following figure shows the state transition matrix for the Multi Lane Maneuver class.

Multi Lane Maneuver State Table

	External	Cannot complete	Internal	Start maneuver	Abort requested	Lane change in progress	Success	Lane changed	Already there
Context states									
CHANGING DRIVING LANE		Unsuccessful multi lane maneuver		CH-BSG	CH-BSG	CH-BSG	CH-BSG	Initialize next maneuver	CH-BSG
Transitory states									
Set maneuver direction		CH-BEE		Initialize next maneuver	CH-BSG	CH-BSG	CH-BSG	CH-BSG	Successful multi lane maneuver
Initialize next maneuver		CH-BEE		CH-BSG	Unsuccessful multi lane maneuver	CHANGING DRIVING LANE	Successful multi lane maneuver	CH-BSG	CH-BSG
Final Deletion states									
Unsuccessful multi lane maneuver		CH-DEL		CH-DEL	CH-DEL	CH-DEL	CH-DEL	CH-DEL	CH-DEL
Successful multi lane maneuver		CH-DEL		CH-DEL	CH-DEL	CH-DEL	CH-DEL	CH-DEL	CH-DEL

Multi Lane Maneuver State Table / tri.guidance.ego.td.18b / August 29, 2021 / Version 1.0.0 / Leon Starr / TRI Systems Engineering Library

Copyright 2021 Toyota Research Institute

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Figure 3.4: Multi Lane Maneuver State Transition Table

Comments on non-transition event responses in the state transition table

Each cell in the table represents an instance's response to an event as a transition (green), ignore (blue) or can't happen (red).

A transition response will appear on the state machine diagram.

An ignore response indicates that if the event occurs in the given state it will be ignored. This is not an error, but a normal behavior. The explanation of why this is a valid response is tagged in the cell as IGN<number> and explained in the associated table of non-transition response comments below.

A can't happen response indicates that our model is broken if that event actually does occur. When 'can't happens' occur in running code, we need to go back and debug our models. The explanation of why this is a valid response is tagged in the cell as CH<number> and explained in the associated table of non-transition response comments below.

Comment	Description
IGN-1	
CH-1	
CH-BSG	Can't happen (Blind to Self Generated event from other state) Self generated event in other state cannot be seen here.
CH-DEL	Can't happen, the instance is deleted in this state as soon as the state's activity has completed
CH-BEE	Blind to External Events. This transient state reacts to its own self generated event only. So it should never see a non-self generated event.

States

The third facet of a complete Executable UML model is a set of activities. An activity may perform a computation, access attributes, manage temporary variables, communicate with external entities and send immediate or delayed signals to other state machine instances. Every state defines a state activity which consists of a set of zero or more actions. Therefore, some states have no actions at all and simply represent a context.

Context states

States named with all uppercase letters represent a period of time where an instance waits for some delayed (time based) or external event to arrive. Such a state represents the current context of the instance.

CHANGING DRIVING LANE

What's going on in this state

A Driving Lane Change is in progress

Informal action taken

No actions.

Transitory states

A state is transitory if it is exited as soon as its activity completes on some event that the instance sends to itself. In Executable UML any self directed event is processed ahead of any pending external event. This makes it possible, for example, to perform an if-then evaluation and then exit on either condition with the appropriate self directed event without the possibility of external interference.

Initialize next maneuver

What's going on in this state

Choose the nearest inner or outer driving lane as the target of a single driving lane change. If there are no more lane changes to perform, complete this maneuver

Informal action taken

First we check to see if we are now in the target lane. If so, we end the maneuver. Otherwise, we check to see if there is a request to abort the maneuver. If so, we declare an unsuccessful maneuver and quit. Otherwise, we request the next Driving Lane Change and wait for it to complete.

Action

```
// Choose the nearest inner or outer driving lane as the target of a driving ↩
     lane change
// Get the Ego's current Driving Lane and the target Driving Lane
ego dl .= /R157/R151/Driving Lane
target dl .= /R157/Driving Lane
// Whether or not an abort was requested, report success if if we ended up  ↩
    in the target EL
(ego dl == target dl) ? Success -> me :
    // If an abort is requested, report failure
    Abort requested? Abort requested -> me : {
        // Initiate the next lane change
        active_lcs .= /R157/EVTRAV.LCS // In a future version we should  ↩
            check to verify that we obtain the instance
        next dl .= ego dl.Adjacent DL( Direction, lcs: active_lcs )
        next dl? {
            Shift lane -> *Driving Lane Change( ) &R152 next dl
            Lane change in process -> me
        } : Abort requested -> me
    }
```

Set maneuver direction

What's going on in this state

Figure out which direction to go to get to the target lane. Is it to the inside or outside?

Informal action taken

We compare the Ego's current inside lane division position with the target inside lane division position. Since the outermost position is 0, increasing in distance toward the inside, we are changing toward the inside if the number is greater, otherwise to the outside. If they are equal then we just quit and declare success.

Action

```
// Is the target driving lane to the inside or outside of the Ego's
//  current lane? We compare the Ego's current inside lane division
//  position with the target inside lane division position. Since
//  the outermost position is 0, increasing in distance toward the
//  inside, we are changing toward the inside if the number is
//  greater, otherwise to the outside.  If they are equal then we
//  just quit and declare success.

ego dl .= /R157/Traveling Ego Vehicle/R151/Driving Lane
ego pos .= ego dl/R10/R19/R21/is the inside boundary of lane/Lane Division. ↩
    Position
```

```
target dl .= /R157/Driving Lane
target pos .= target dl/R10/R19/R21/is the inside boundary of lane/Lane  ↩
    Division.Position

target  pos == ego pos? Already there -> me : {
    target pos > ego  pos ? Direction = .outer : .inner
    Start maneuver -> me
}
```

Final deletion states

In final deletion states, the class instance are automatically deleted after the activity is performed.

Successful multi lane maneuver

What's going on in this state

All the necessary Driving Lane Changes have completed successfully and we are now in the target lane. We set the final turn signal condition as requested at the beginning of this multi-lane maneuver.

Informal action taken

When the MLM was first requested, a desired end condition turn signal was requested. This may mean that we cancel the signal upon completion or that we leave it on in the direction of an intersection turn. (Which may or may not match the direction of the MLM! Consider shifting into the left most right turn lane, for example)

Action

```
ELA.Successful_mult_lane_change() // Just a stub for now
// ELA = Entrance Lane Approach class/relationship be
// added when the intersection subsystem is released
//
PANEL.Indicate ( direction: Turn direction )
```

Unsuccessful multi lane maneuver

What's going on in this state

We did not make it into the target lane and we've given up

Informal action taken

The ELA (Entrance Lane Approach) is notified that the MLM has failed. NOTE: ELA = Entrance Lane Approach which is a relationship/class in the intersection subsystem, not yet but soon to be released.

Action

```
ELA.Unsuccessful_mult_lane_change()
```

Chapter 4

Processing Model

The third facet of the xUML models describes the algorithmic processing of the activities in states, methods, domain operations and external entity operations. All activities are described using the Scrall action language.

Since state activities are written inside the activity compartments of states on the state machine diagram, they have already been covered in the behavior section along with the state models.

In this chapter, we include the action language for non-state activities.

Domain Operations

A model level "API" is defined as a set of domain operations. Each operation provides opaque access preventing any external domains or code from needging to know what specific model elements, if any, lie inside the encapsulated domain.

When invoked, a domain operation will map the provided input to corresponding internal semantics and direct that information to the relevant internal elements for processing. A domain operation is defined as synchronous or asynchronous. If it is synchronous, a value of arbitrary complexity may be returned as part of the operation invocation in a manner similar to calling a programming language function. If the operation is asynchronous, the invocation completes with no value returned, but activity is triggered inside the target domain which may result in some future response or other effect.

Domain Operations are defined on an entire domain and not by subsystem. Therefore those defined in this section, are available to all supbsystems of this Vehicle Guidance domain.

Init

The **Init** operation is a special operation defined on each domain by the model execution environment. It is expected to be called sometime during system startup. This is typically done as part of `main()` before the event loop is entered. The model execution environment also accommodates subsystem specific *Init* calls that are triggered by the domain level **Init**, however none are required at this point since a single subsystem comprises our domain.

Abort lane change

The **Abort lane change** operation discontinues an active lane change maneuver.

Action

```
// Domain operation
// ---
// Abort lane change()
// ---

dl = Driving Lane Change(1)
dl? Abort -> dl
```

Crossing completed

The **Crossing completed** operation informs the Ego Vehicle subsystem that the vehicle has completely crossed into a new lane.

Action

```
// Domain operation
// ---
// Crossing completed()
// ---

dl = Driving Lane Change(1)
dl? Crossing completed -> dl
```

Crossing lane division

The **Crossing lane division** operation informs the Ego Vehicle subsystem that the vehicle has begun crossing out of its lane.

Action

```
// Domain operation
// ---
// Crossing lane division()
// ---

dl = Driving Lane Change(1)
dl? Crossing -> dl
```

Ego arrived in lane

The **Ego arrived in lane** operation determines if the Ego Vehicle has successfully arrived into the previously requested lane.

Action

```
// Domain operation
// ---
// Ego arrived in lane( inside division: Lane Division ID, outside division: ↩
     Lane Division ID,
                         road segment: Road Segment ID )
//
ev = Traveling Ego Vehicle(1)
ev? {
    Traveling Ego Vehicle(1).New lane( in.inside division, in.outside  ↩
        division, in.road segment )
    =>> True
} : =>> False


// If the ego isn't traveling, there's a major problem.  False indicates an  ↩
    error back to the
// calling EE
```

Get into lane

The **Get into lane** operation specifies the intended target lane and initiates a Multi Lane Maneuver.

Action

```
// Domain operation
// ---
// Get into lane( completion turn signal: Turn Signal, inside division: Lane ↩
     Number
//     outside division: Lane Number ) : Boolean
// ---
traveling ev = Traveling Ego Vehicle (1)

target dl = Driving Lane( in.inside division, in.outside division,
                          traveling ev/R151/Driving Lane.Road segment )

target dl? // If found, send MLM creation event and return true, else return ↩
     false
    Get into lane -> *Multi Lane Maneuver(
        in.completion turn signal ) &R157 target dl, traveling ev ; =>> True ↩
             : =>> False
```

Request MLM abort

The **Request MLM abort** operation discontinues a Multi Lane Maneuver.

Action

```
// Domain operation
// ---
```

```
// Request MLM abort()
//
active_mlm = MLM(1) // Select the one and only MLM instance
active_mlm? // If it exists
    // Send an event, set a boolean attribute to True, return True value
    Abort requested -> active_mlm ; active_mlm.Abort requested.set(); =>>  ↩
        True :
    // ; means that there is no data dependency among actions, so you can do ↩
         them in any order
    =>> False // It doesn't exist, return the False value
```

Target lane status

The **Target lane status** operation is set to open or closed to indicate whether or not it is currently possible to change into the desired lane.

Action

```
// Domain operation
// ---
// Target lane status ( open: Boolean )
//

dlc = Driving Lane Change(1)
dlc? {
    dlc.Lane status( in.open ) =>> True : =>> False
}
```

Methods

A method is an activity defined on a class and is executable by any instance of that class.

Driving Lane Change Methods

Lane status

The **Lane status** method is used to determine whether the target lane is open or closed.

Action

```
// Class method: Driving Lane Change
// ---
// Lane status( status )
// ---

// Set the Target lane open attribute to true or false
// and fire off an event to the DLC instance if the incoming status is
```

```
// different than the Target lane open value.  In other words, don't do  ↩
    anything
// unless there is a change in value

status != Target lane open? {
    Target lane open = status
    status? Target lane open -> : Target lane closed -> me
}
```

Start monitoring target lane

The **Start monitoring target lane** method notifies the LANE MONITOR external entity that it should start monitoring the target lane and report its open/close status. Returns an immediate true/false evaluation as to whether or not the target lane is currently open (true) or closed.

Action

```
// Class method: Driving Lane Change
// ---
// Start monitoring target lane() : Boolean
// ---

=>> LANE MONITOR.Target lane designated(
    inside division: Target lane inside division,
    outside division: Target lane outside division,
    Road segment
)
```

Stop monitoring target lane

The **Start monitoring target lane** method notifies the LANE MONITOR external entity that it should stop monitoring the target lane and not report its open/close status. No value returned.

Action

```
// Class method: Driving Lane Change
// ---
// Stop monitoring target lane()
// ---

LANE MONITOR.Target lane released(
    inside division: Target lane inside division,
    outside division: Target lane outside division,
    Road segment
)
```

Traveling Ego Vehicle Methods

New lane

Action

```
// Class method: New Lane
// ---
// New lane( inside: Lane Division ID, outside: Lane Division ID,
//  road segment: Road Segment ID )
// ---
//

// Look up the new Driving Lane instance based in parameter input
new dl .= Driving Lane( Inside division: in.inside,
                        Outside division: in.outside,
                        Road segment: in.road segment )

// Get the current ego lane (there must be one)
active dl .= /R151/Active Driving Lane

// If it is the same lane, there's nothing to do
// Scrall implies that you compare the id values of each instance
// to determine whether are not they match)
active dl != new dl? {
    &R151 new dl, active dl < 1 >   // We set a control token since sequence
                                    // is important here < 1 > dlc in
                                    // progress .= /R151/R152/Driving Lane
                                    // Change // When < 1 > is set, this
                                    // action may execute

    dlc in progress?                // There may or may not be one at any
                                    // given moment
        Crossing completed -> dlc in progress
}

//
```

External Entities

An External Entity is a proxy for some component that lies external to our domain. In many cases, the external component will be another complete domain, modeled or otherwise. But it is often helpful to use a more useful local label especially if the external domain has not yet been defined or if a certain label is more meaningful with respect to our domain.

External entities represent the detail control and data interfaces that press requirements on other parts of the system. Bridging code will map the requirements represented by the external entities onto services provided by other domains in the system. We then have the convenience of using locally meaningful terminology

consistent with our domain's semantics in defining interactions with an external component. It is the responsibility of the bridging code to map the semantics of the Vehicle Guidance onto the semantics of services provided by other domains.

For brevity, the following external entity interfaces assume that an identifier of the instance invoking the entity is available to the processing of the entity. You may think of external entity operation as similar to an instance operation in that there is an implied parameter of the identifier of the invoking instance. Consequently, the identifier of the invoker is not shown below[1].

DRIVING External Entity

The `DRIVING` external entity represents whoever or whatever drives the ego vehicle. This is not to be confused with steering and torques which are at a finer level of control. It may help to imagine `DRIVING` as a student driver with Vehicle Guidance as the driving instructor (though we aren't trying to teach!). Or maybe a nervous parent is a better representation of our role with respect to `DRIVING`. Thus interactions are along the lines of "Go through this signal", "Wait at this signal", "hold at this crosswalk", "Get in the outside lane as you make this left turn", "hold here and let the oncoming traffic pass, okay now go!" That sort of thing. We assume that `DRIVING` can make turns, creep along through traffic and manage brakes and steering competently.

The following `DRIVING` external entity operations are assumed.

```
Cancel maneuver to target lane
```

Operation: Cancel maneuver to target lane()

Reports that the lane change maneuver is being canceled prior to entry into the target lane

```
Incomplete Lane Change
```

Operation: Incomplete lane change()

Reports that the lane change can not be conpleted because the maneuver time has expired or the driver has aborted the maneuver or a crossing was detected after the target lane was acquired, but before the lane change completed.

```
Inhibit preempted
```

Operation: Inhibit preempted()

Reports that the lane change inhibit period was interrupted with a premature successive lane change.

[1] Although it will make a reappearance in the translation.

`Lingering cross`

Operation: Lingering cross()

Reports that the lane division crossing is taking an excessive amount of time since the beginning of the physical maneuver toward the division

`Maneuver to target lane`

Asynchronous: Maneuver to target lane(dir: Left Right)

Indicates that the lane change is ready for the beginning of the maneuver into the target lane in the indicated direction.

`Max lane change time exceeded`

Operation: Max lane change time exceeded()

Reports that the maximum duration of an entire lane change behavior has been exceeded.

`Post crossing abort`

Operation: Post crossing abort()

An abort event was detected after successful occupation of the Target Lane

`Return to source lane`

Asynchronous: Return to source lane(Inside division: Lane Division, Outside division: Lane Division road segment: Nominal)

Indicates that the lane change is ready for the beginning of the maneuver into the target lane in the indicated direction.

`Target lane unavailable`

Operation: Target lane unavailable()

Reports that the lane change is aborted because too much time has passed without an available opening in the target lane.

`Unexpected crossing after lane change`

Operation: Unexpected crossing after lane change()

During post indication (signals still on after occupying the target lane) a Lane Division was crossed. This action fails and terminates the lane behavior.

`Unexpected crossing during successive lane change inhibition period`

Operation: Unexpected crossing during successive lane change inhibition period()

While inhibiting the next lane change, a crossing event was detected. This condition fails and terminates the lane change behavior.

`Unsafe crossing`

Operation: Unsafe crossing() Reports that the lane change is now unsafe due to a premature lane crossing.

LANE MONITOR External Entity

The Vehicle Guidance requires a LANE MONITOR entity to continuously observe the target lane looking for an open space. This external entity considers the Ego Vehicle's speed and acceleration as well as that of any other traffic participants directly influencing the opportunity to move into an adjacent lane. It then lets us know when the Ego Vehicle can or cannot safely attempt entry into the designated target lane.

The following `LANE MONITOR` external entity operations are assumed.

`Target lane designated`

Operation: Target lane designated(inside_division: Lane Division, outside_division: Lane Division, road_segment: Road Segment)

When a target lane has been designated, begin monitoring it for an open space lane change opportunity.

`Target lane released`

Operation: Target lane released(inside_division: Lane Division, outside_division: Lane Division, road_segment: Road Segment)

When a target lane has been released, stop monitoring it for an open space lane change opportunity.

`Transition ahead`

Operation: Transition ahead(distance : Distance, traversable : Boolean, cross ok : Boolean)

When a target lane has been released, stop monitoring it for an open space lane change opportunity

PANEL External Entity

The Vehicle Guidance requires the ability to indicate the intention to make a lane change to nearby traffic participants. We assume the existence of a PANEL external entity that provides access to the turn indicator. This term is used to generally refer to the dashboard or driver panel controls, though in this case, we are concerned only about turn signals.

```
Indicate
```

Operation: Indicate(direction : Turn Signal)

Tells the instrument panel / steering column to signal a left or right turn or to cancel the active turn signal, if any

Entrance Lane Approach Stubs

To support multi-lane lane changes.

```
Successful multi lane change
```

Operation: Successful multi lane change()

The multi lane change maneuver has succeded

```
Unsuccessful multi lane change
```

Operation: Unsuccessful multi lane change()

The multi lane change maneuver has failed

Part I

Reference Material

Bibliography

Books

[1] [mb-xuml] Stephen J. Mellor and Marc J. Balcer, Executable UML: a foundation for model-driven architecture, Addison-Wesley (2002), ISBN 0-201-74804-5.

[2] [rs-xuml] Chris Raistrick, Paul Francis, John Wright, Colin Carter and Ian Wilkie, Model Driven Architecture with Executable UML, Cambridge University Press (2004), ISBN 0-521-53771-1.

[3] [mtoc] Leon Starr, Andrew Mangogna and Stephen Mellor, Models to Code: With No Mysterious Gaps, Apress (2017), ISBN 978-1-4842-2216-4

[4] [ls-build], Leon Starr, How to Build Shlaer-Mellor Object Models, Yourdon Press (1996), ISBN 0-13-207663-2.

[5] [sm-data] Sally Shlaer and Stephen J. Mellor, Object Oriented Systems Analysis: Modeling the World in Data, Prentice-Hall (1988), ISBN 0-13-629023-X.

[6] [sm-states] Sally Shlaer and Stephen J. Mellor, Object Oriented Systems Analysis: Modeling the World in States, Prentice-Hall (1992), ISBN 0-13-629940-7.

Articles

[7] [ls-articulate] Leon Starr, How to Build Articulate UML Class Models, 2008, http://www.modelint.com/how-to-build-articulate-uml-class-models/

[8] [ls-scrall] Leon Starr, Scrall Action Language Specification, https://github.com/modelint/-scrall

[9] [ls-time] Leon Starr, Time and Synchronization in Executable UML, 2008, https://www.modelint.com/time-and-synchronization-in-executable-uml/

Index