# Torpedo Launch Domain Workbook

## A Pycca Translation

**REVISION HISTORY**

| NUMBER | DATE | DESCRIPTION | NAME |
|---|---|---|---|
| 1.0 | January 17, 2017 | Initial draft. | GAM |
| 1.1 | March 11, 2017 | Clean up for release. | GAM |

# Contents

## List of Figures

# Introduction

This document describes a torpedo launch domain. The domain presented here is an example that is contained in the book, *Models To Code*. This example is *not* completely worked out. In the book, only part of the domain is given and we are supplying only some of the missing details here.

## Literate Program

This document is also a literate program which means that it contains both the descriptive material for the domain as well as the source code that implements the domain. A document, in many different formats (*e.g.* PDF), can be generated from the source using `asciidoc`[1]. The source file is a valid `asciidoc` file. The `pycca` source code that implements the model can be extracted from the document using a literate programming tool named, `atangle`[2]. The syntax of the literate program *chunks* is given below.

**Important**
This literate program document intermixes model elements and implementation elements. This makes the correspondence between the model and its translation very clear and simplifies propagating model changes into the translation. However, we must be clear that producing the domain is a two step process. The model **must** be fully complete before the translation can be derived from it. Do not let the appearance of the final document suggest that the process of obtaining it was one of incremental refinement or that translating the model took place at the same time as formulating it. Just as one does not write a novel from beginning to end in a single pass, one does not construct a domain in the same order as it is presented here.

# Data Types

The domain defines several model level data types.

## Torpedo ID

A Torpedo ID is the set of values used to identify individual torpedoes.

**Torpedo_ID Implementation**

```
<<external data types>>=
typedef uint8_t Torpedo_ID ;
```

## Launch type

There are two ways that torpedoes are launched and the Launch type defines those two techniques.

**Launch Type Implementation**

```
<<internal data types>>=
typedef enum {
    Passive,
    Active
} L_type ;
```

[1] http://www.methods.co.nz/asciidoc/
[2] http://repos.modelrealization.com/cgi-bin/fossil/mrtools

### Tube number

Torpedo tubes are numbered with small sequential numbers.

**Tube Number Implementation**

```
<<internal data types>>=
typedef uint8_t Tube_number ;
```

## External Operations

Torpedo motor operations are delegated to the MOTOR external entity.

### MOTOR.start

The MOTOR.start operation is invoked when a passively Launched Torpedo has been expelled from its tube and can then be deemed as a Fired Torpedo.

**torpedo**
Identifier of the torpedo whose motor is to be started.

**MOTOR start Implementation**

```
<<external operations>>=
external operation
MOTOR_start(
    Torpedo_ID torpedo)
{
}
```

## Classes and Relationships

Discuss the class diagram. The preferred method is to discuss classes and relationships in a natural order rather than separating classes and relationships into separate sections. This is in better keeping with the class diagram

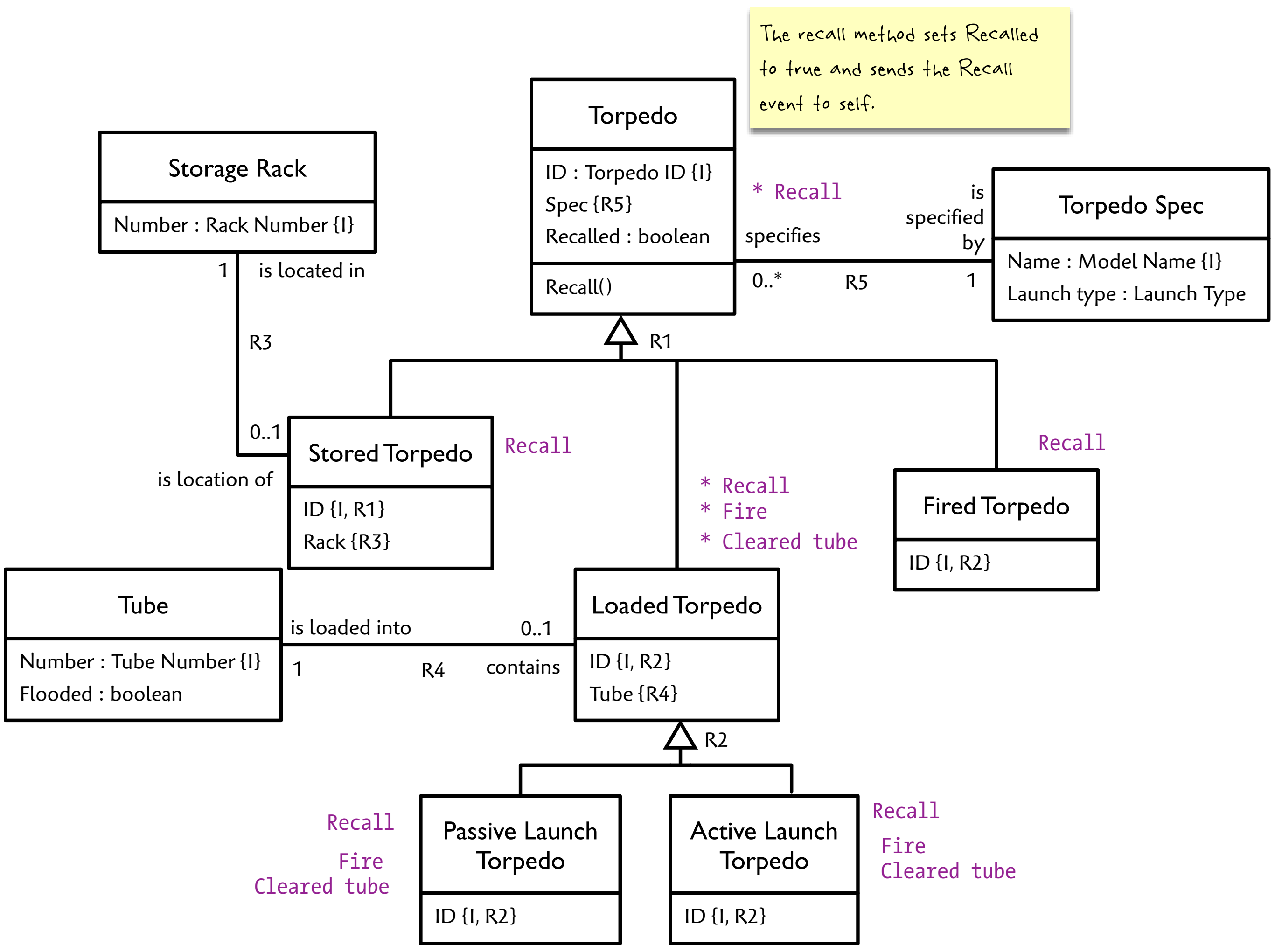

Figure 1: Torpedo Launch Class Diagram

## Torpedo

A Torpedo is a cigar-shaped missile containing explosives that may be launched in order to destroy other vessels. We will only consider those Torpedoes that are launched from a submarine.

### Attributes

**ID {I}**
Identifier.

**Spec {R5}**
Torpedo specification name.

**Recalled**
Boolean to register recall request.

### Implementation

```
<<torpedo attributes>>=
attribute (Torpedo_ID ID)
attribute (bool Recalled) default {false}
```

#### Instance Operations

#### Recall

#### Recall Activity

```
set Recalled
Recall => me
```

#### Recall Implementation

```
<<torpedo instance operations>>=
instance operation
Recall()
{
    self->Recalled = true ;
    PYCCA_generatePolymorphic(Recall, Torpedo, self, self) ;
}
```

## Torpedo Spec

A Torpedo Spec defines the characteristics of Torpedoes.

### Attributes

**Name {I}**
Identifier.

**Launch type**
A values to determine the manner in which a Torpedo is launched.

#### Implementation

```
<<torpedo spec attributes>>=
attribute (char const * Name)
attribute (L_type Launch_type)
```

## R5

- Torpedo is specified by *exactly one* Torpedo Spec
- Torpedo Spec specifies *zero or more* Torpedo

A Torpedo is described by a specification. We allow for Torpedo Specifications to exist that may not describe any Torpedo in our system.

#### R5 Implementation

```
<<torpedo references>>=
reference R5 -> Torpedo_Spec
```

### R1

- Torpedo *is a* Stored Torpedo or Loaded Tropedo or Fired Torpedo

A Torpedo may by in one of three conditions. It is either stored on a rack, loaded into a launch tube or it has been fired and is on its way to the target.

```
<<torpedo references>>=
polymorphic event
    Recall
end

subtype R1 reference
    Stored_Torpedo
    Loaded_Torpedo
    Fired_Torpedo
end

<<stored torpedo references>>=
reference R1 -> Torpedo

<<loaded torpedo references>>=
reference R1 -> Torpedo

<<fired torpedo references>>=
reference R1 -> Torpedo
```

## Stored Torpedo

A Stored Torpedo is that type of Torpedo that is currently sitting in a rack awaiting deployment.

#### Attributes

**ID {I,R1}**
Identifier.

**Rack {R3}**
Storage rack location.

## Storage Rack

A Storage Rack is a place where Stored Torpedoes reside.

#### Attributes

**Number {I}**
Identifier.

#### Implementation

```
<<storage rack attributes>>=
attribute (unsigned Number)
```

### R3

- Stored Torpedo is located in *exactly one* Storage Rack
- Storage Rack is the location of *zero or one* Stored Torpedo

In order to launch a torpedo, we must know its location in a Storage Rack. No more than one Torpedo may be stored in a given rack location and we allow for a rack location to be empty.

**R3 Implementation**

```
<<stored torpedo references>>=
reference R3 -> Storage_Rack
```

## Loaded Torpedo

A Loaded Torpedo is that type of Torpedo which is currently residing in launch tube and thus can be launched.

#### Attributes

**ID {I,R1}**
Identifier.

**Tube {R4}**
Torpedo tube in which torpedo is loaded.

## Tube

A Tube is a location in the submarine from which a Torpedo may be launched.

#### Attributes

**Number {I}**
Identifier.

**Flooded**
Boolean to tube condition.

**Implementation**

```
<<tube attributes>>=
attribute (Tube_number Number)
attribute (bool Flooded) default {false}
```

### R4

- Loaded Torpedo is loaded into *exactly one* Tube
- Tube contains *zero or one* Loaded Torpedo

In order to launch a Torpedo it must be in a Tube and we launch Torpedoes from a given Tube. We also allow for the a Tube to be empty.

**R5 Implementation**

```
<<loaded torpedo references>>=
reference R4 -> Tube
```

### Fired Torpedo

A Fired Torpedo is that type of Torpedo which is on it way to its target.

### Passive Launch Torpedo

A Passive Launch Torpedo is that type of Torpedo which is launched from a tube by a combination of water and air pressure.

#### Attributes

**ID {I,R1}**
Identifier.

### Active Launch Tropedo

An Active Launch Tropedo is that type of Torpedo which uses a self-contained motor to launch itself from a tube.

#### Attributes

**ID {I,R1}**
Identifier.

### R2

- Loaded Torpedo *is a* Passive Launch Torpedo or Active Launch Torpedo

Our submarine supports launching either type of torpedo design.

```
<<loaded torpedo references>>=
polymorphic event
    Fire
    Cleared_tube
end

subtype R2 reference
    Passive_Launch_Torpedo
    Active_Launch_Torpedo
end

<<passive launch torpedo references>>=
reference R2 -> Loaded_Torpedo

<<active launch torpedo references>>=
reference R2 -> Loaded_Torpedo
```

# Active Classes

## Stored Torpedo State Model

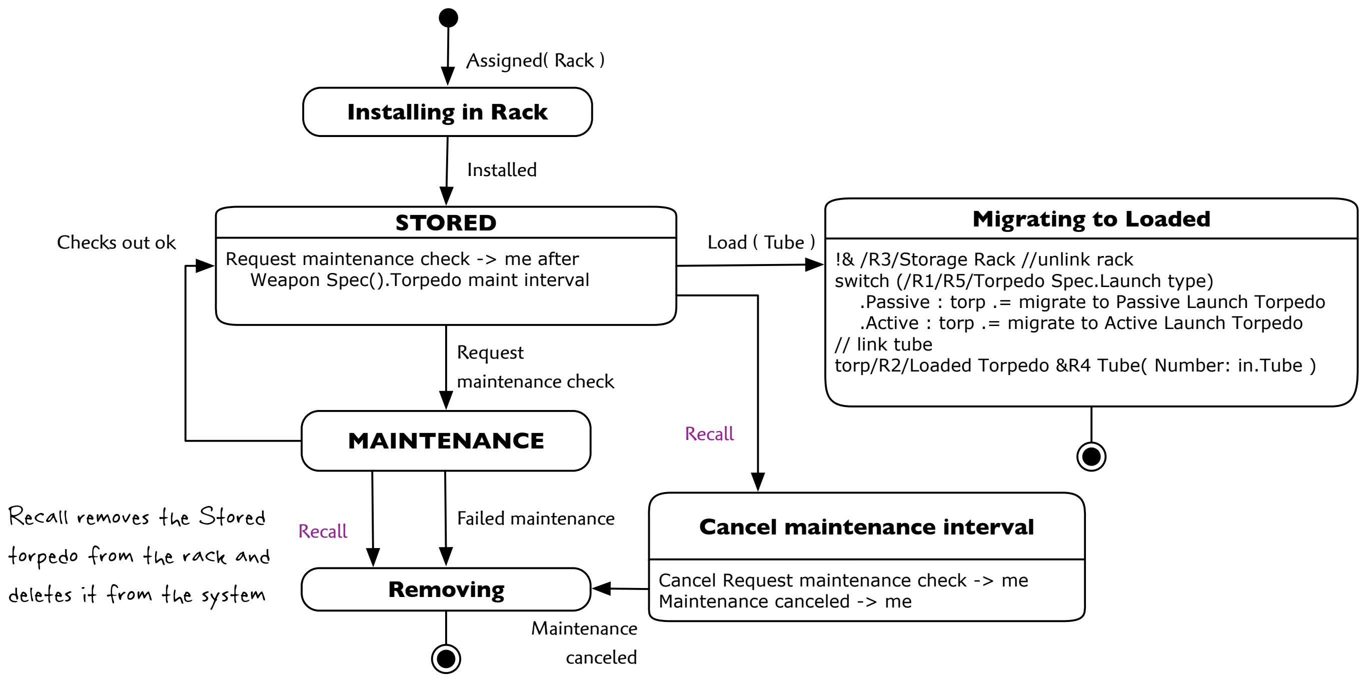
Assigned( Rack )
Installing in Rack
Installed
STORED
Request maintenance check -> me after
Weapon Spec().Torpedo maint interval
Checks out ok
Load ( Tube )
Migrating to Loaded
!& /R3/Storage Rack //unlink rack
switch (/R1/R5/Torpedo Spec.Launch type)
.Passive : torp .= migrate to Passive Launch Torpedo
.Active : torp .= migrate to Active Launch Torpedo
// link tube
torp/R2/Loaded Torpedo &R4 Tube( Number: in.Tube )
Request
maintenance check
MAINTENANCE
Recall
Recall removes the Stored torpedo from the rack and deletes it from the system
Recall
Failed maintenance
Cancel maintenance interval
Cancel Request maintenance check -> me
Maintenance canceled -> me
Removing
Maintenance
canceled

Figure 2: Stored Torpedo State Model

### Installing in Rack

```
Place action language for the state here.
```

### Installing in Rack Implementation

```
<<stored torpedo state model>>=
state Installing_in_Rack(uint8_t rack) {
    // Include your C code for the state here.
}
transition . - Assigned -> Installing_in_Rack
transition Installing_in_Rack - Installed -> STORED
```

### STORED

```
Request maintenance check -> me after
Weapon Spec().Torpedo maint interval
```

### STORED Implementation

```
<<stored torpedo state model>>=
state STORED() {
    // Include your C code for the state here.
}
transition STORED - Load -> Migrating_to_Loaded
transition STORED - Recall -> Cancel_maintenance_interval
transition STORED - Request_maintenance_check -> MAINTENANCE
```

### MAINTENANCE

```
Place action language for the state here.
```

### MAINTENANCE Implementation

```
<<stored torpedo state model>>=
state MAINTENANCE() {
    // Include your C code for the state here.
}
transition MAINTENANCE - Checks_out_ok -> STORED
transition MAINTENANCE - Failed_maintenance -> Removing
transition MAINTENANCE - Recall -> Removing
```

### Removing

```
Place action language for the state here.
```

### Removing Implementation

```
<<stored torpedo state model>>=
state Removing() {
    // Include your C code for the state here.
}
final state Removing
```

### Migrating to Loaded

```
!& /R3/Storage Rack //unlink rack
switch (/R1/R5/Torpedo Spec.Launch type)
.Passive : torp .= migrate to Passive Launch Torpedo
.Active : torp .= migrate to Active Launch Torpedo
// link tube
torp/R2/Loaded Torpedo &R4 Tube( Number: in.Tube )
```

### Migrating to Loaded Implementation

```
<<stored torpedo state model>>=
state Migrating_to_Loaded(
    Tube_number Number) {

        // Get a reference to the superclass instance.
    ClassRefVar(Torpedo, torp) = self->R1 ;
        // Create an instance of Loaded Torpedo
    ClassRefVar(Loaded_Torpedo, ltorp) = PYCCA_newInstance(Loaded_Torpedo) ;
        // Relate the Loaded Torpedo to the Torpedo across R1.
    PYCCA_relateSubtypeByRef(torp, Torpedo, R1, ltorp, Loaded_Torpedo) ;
    ltorp->R1 = torp ;
        // Relate the Loaded Torpedo to the Tube.
    ltorp->R4 = PYCCA_refOfId(Tube, rcvd_evt->Number) ;

        // Now handle R2
    switch (torp->R5->Launch_type) {
        case Active: {
                // For Active Launch Torpedos, create a new instance.
            ClassRefVar(Active_Launch_Torpedo, altorp) =
                PYCCA_newInstance(Active_Launch_Torpedo) ;
                // Relate the Active Launch Torpedo across R2.
            PYCCA_relateSubtypeByRef(ltorp, Loaded_Torpedo, R2, altorp,
                    Active_Launch_Torpedo) ;
            altorp->R2 = ltorp ;
        }
        break ;
        case Passive: {
                // For Passive Launch Torpedos, create a new instance.
            ClassRefVar(Passive_Launch_Torpedo, pltorp) =
                PYCCA_newInstance(Passive_Launch_Torpedo) ;
                // Relate the Passive Launch Torpedo across R2.
            PYCCA_relateSubtypeByRef(ltorp, Loaded_Torpedo, R2, pltorp,
                    Passive_Launch_Torpedo) ;
            pltorp->R2 = ltorp ;
        }
        break ;
    }

    // Since this is final state, the Stored Torpedo instance is
    // deleted and since there were no references in Storage Rack back
    // to the Stored Torpedo, we don’t have to destroy the instance
    // of the R3 association.
}
final state Migrating_to_Loaded
```

### Cancel maintenance interval

```
Cancel Request maintenance check -> me
Maintenance canceled -> me
```

### Cancel maintenance interval Implementation

```
<<stored torpedo state model>>=
state Cancel_maintenance_interval() {
    // Include your C code for the state here.
}
transition Cancel_maintenance_interval - Maintence_canceled -> Removing
```

## Passive Launch Torpedo State Model

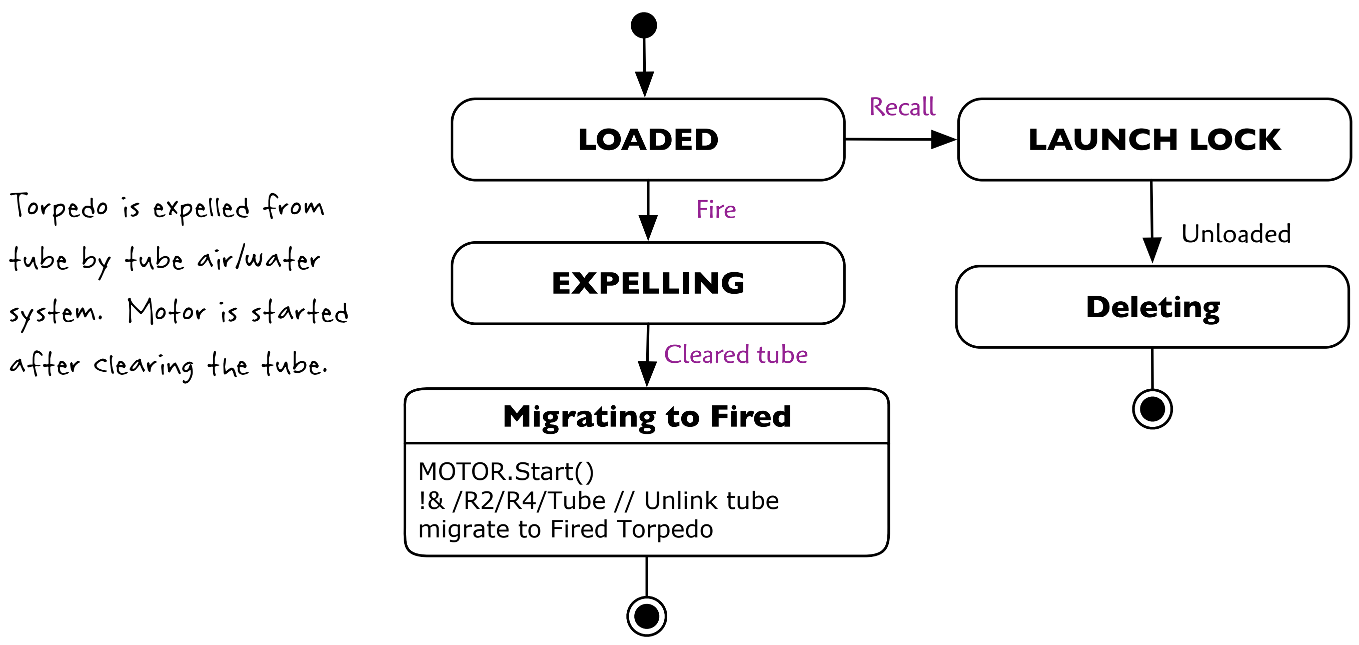
Torpedo is expelled from tube by tube air/water system. Motor is started after clearing the tube.
LOADED
Recall
LAUNCH LOCK
Fire
Unloaded
EXPELLING
Deleting
Cleared tube
Migrating to Fired
MOTOR.Start()
!& /R2/R4/Tube // Unlink tube
migrate to Fired Torpedo

Figure 3: Passive Torpedo State Model

### LOADED

```
Place action language for the state here.
```

### LOADED Implementation

```
<<passive launch torpedo state model>>=
state LOADED() {
    // Include your C code for the state here.
}
transition LOADED - Recall -> LAUNCH_LOCK
transition LOADED - Fire -> EXPELLING
```

### LAUNCH LOCK

```
Place action language for the state here.
```

### LAUNCH LOCK Implementation

```
<<passive launch torpedo state model>>=
state LAUNCH_LOCK() {
    // Include your C code for the state here.
}
transition LAUNCH_LOCK - Unloaded -> Deleting
```

### Deleting

```
Place action language for the state here.
```

### Deleting Implementation

```
<<passive launch torpedo state model>>=
state Deleting() {
    // Include your C code for the state here.
}
final state Deleting
```

### EXPELLING

```
Place action language for the state here.
```

### EXPELLING Implementation

```
<<passive launch torpedo state model>>=
state EXPELLING() {
    // Include your C code for the state here.
}
transition EXPELLING - Cleared_tube -> Migrating_to_Fired
```

### Migrating to Fired

```
MOTOR.Start()
!& /R2/R4/Tube // Unlink tube
migrate to Fired Torpedo
```

### Migrating to Fired Implementation

```
<<passive launch torpedo state model>>=
state Migrating_to_Fired() {
    // Include your C code for the state here.
}

final state Migrating_to_Fired
```

## Fired Torpedo State Model

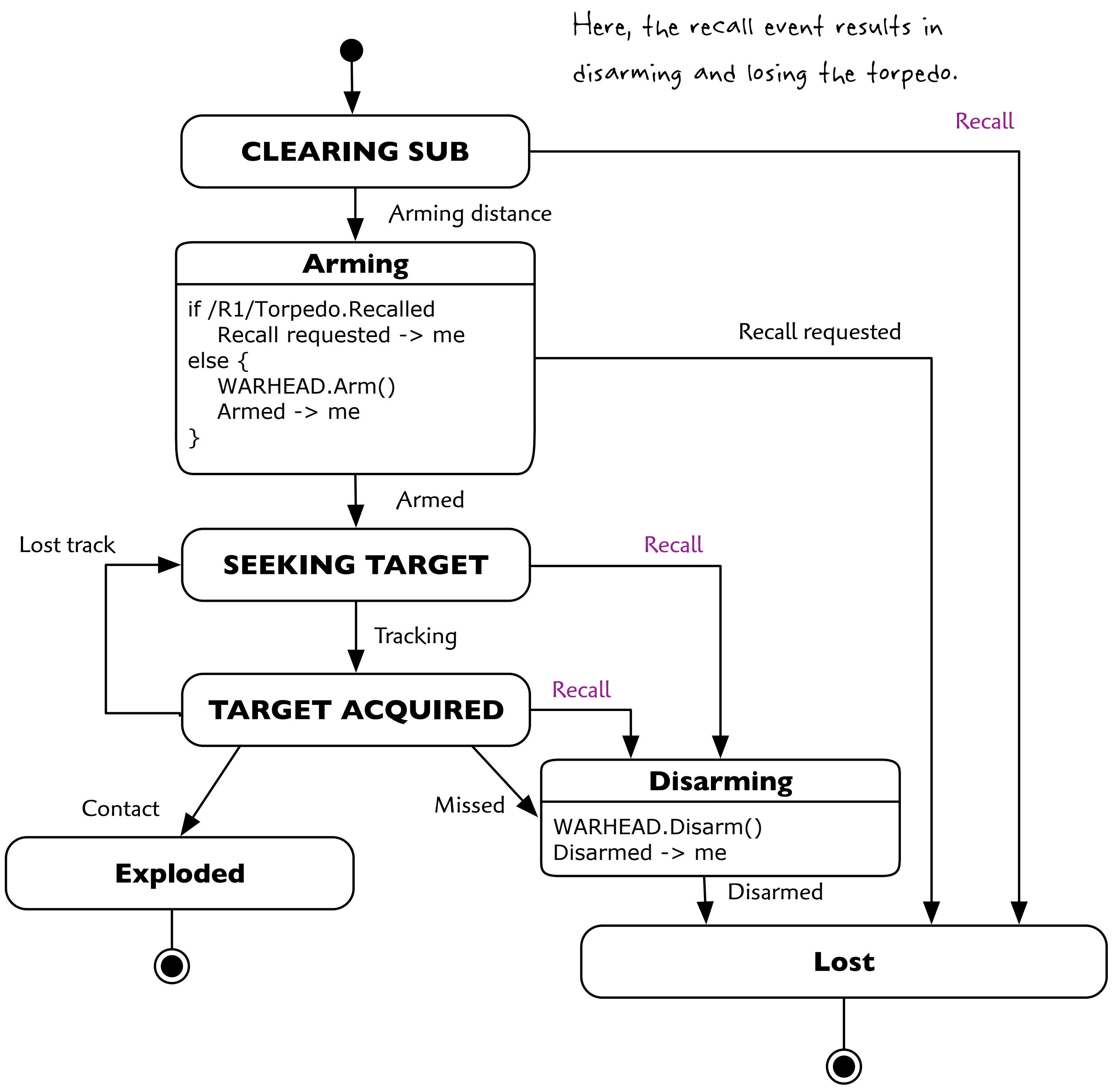

Figure 4: Fired Torpedo State Model

### CLEARING SUB

```
Place action language for the state here.
```

### CLEARING SUB Implementation

```
<<fired torpedo state model>>=
state CLEARING_SUB() {
    // Include your C code for the state here.
}
transition CLEARING_SUB - Recall -> Lost
transition CLEARING_SUB - Cleared_tube -> Arming
```

### Arming

```
if /R1/Torpedo.Recalled
    Recall requested -> me
else {
    WARHEAD.Arm()
    Armed -> me
}
```

### Arming Implementation

```
<<fired torpedo state model>>=
state Arming() {
    // Include your C code for the state here.
}
transition Arming - Recall_requested -> Lost
transition Arming - Armed -> SEEKING_TARGET
```

### SEEKING TARGET

```
Place action language for the state here.
```

### SEEKING TARGET Implementation

```
<<fired torpedo state model>>=
state SEEKING_TARGET() {
    // Include your C code for the state here.
}
transition SEEKING_TARGET - Recall -> Disarming
transition SEEKING_TARGET - Tracking -> TARGET_ACQUIRED
```

### TARGET ACQUIRED

```
Place action language for the state here.
```

### TARGET ACQUIRED Implementation

```
<<fired torpedo state model>>=
state TARGET_ACQUIRED() {
    // Include your C code for the state here.
}
transition TARGET_ACQUIRED - Recall -> Disarming
transition TARGET_ACQUIRED - Missed -> Disarming
transition TARGET_ACQUIRED - Contact -> Exploded
transition TARGET_ACQUIRED - Lost_track -> SEEKING_TARGET
```

### Disarming

```
WARHEAD.Disarm()
Disarmed -> me
```

### Disarming Implementation

```
<<fired torpedo state model>>=
state Disarming() {
    // Include your C code for the state here.
}
transition Disarming - Disarmed -> Lost
```

### Exploded

```
Place action language for the state here.
```

### Exploded Implementation

```
<<fired torpedo state model>>=
state Exploded() {
    // Include your C code for the state here.
}
final state Exploded
```

### Lost

```
Place action language for the state here.
```

### Exploded Implementation

```
<<fired torpedo state model>>=
state Lost() {
    // Include your C code for the state here.
}
final state Lost
```

## Active Launch Torpedo State Model

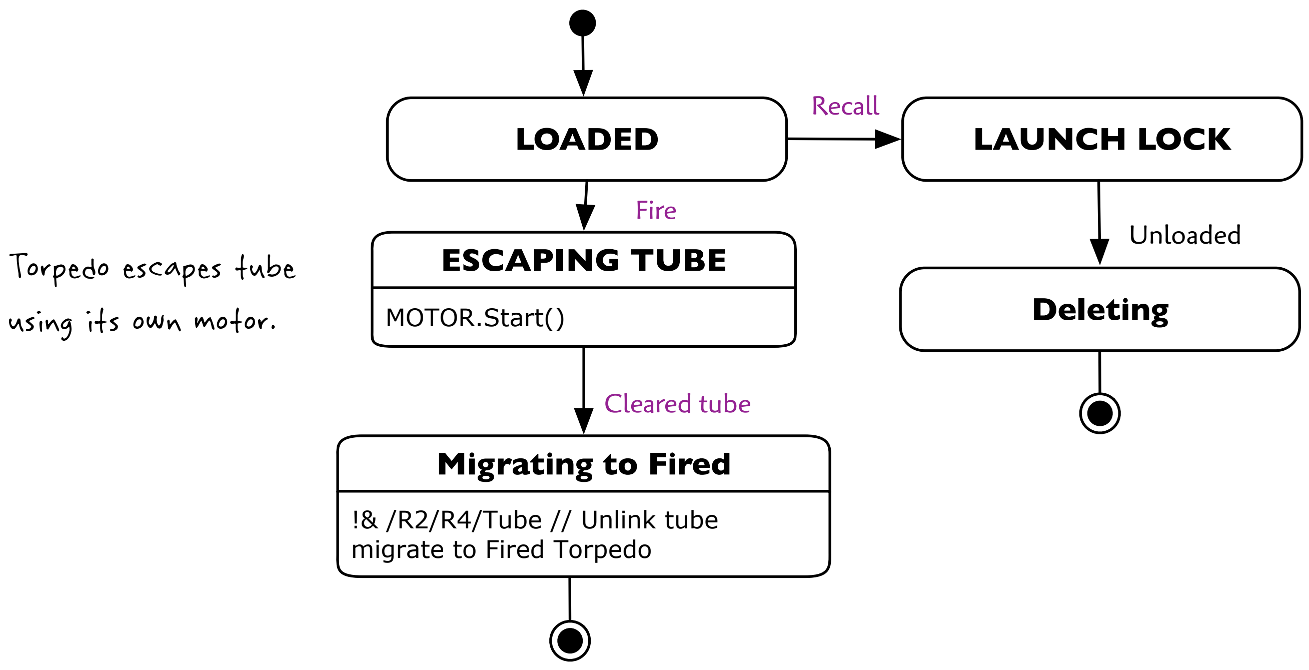

Figure 5: Stored Torpedo State Model

### LOADED

```
Place action language for the state here.
```

### LOADED Implementation

```
<<active launch torpedo state model>>=
state LOADED() {
    // Include your C code for the state here.
}
transition LOADED - Recall -> LAUNCH_LOCK
transition LOADED - Fire -> ESCAPING_TUBE
```

#### LAUNCH LOCK

```
Place action language for the state here.
```

#### LAUNCH LOCK Implementation

```
<<active launch torpedo state model>>=
state LAUNCH_LOCK() {
    // Include your C code for the state here.
}
transition LAUNCH_LOCK - Unloaded -> Deleting
```

#### Deleting

```
Place action language for the state here.
```

#### Deleting Implementation

```
<<active launch torpedo state model>>=
state Deleting() {
    // Include your C code for the state here.
}
final state Deleting
```

#### ESCAPING TUBE

```
MOTOR.Start()
```

#### ESCAPING TUBE Implementation

```
<<active launch torpedo state model>>=
state ESCAPING_TUBE() {
    // Include your C code for the state here.
}
transition ESCAPING_TUBE - Cleared_tube -> Migrating_to_Fired
```

#### Migrating to Fired

```
!& /R2/R4/Tube // Unlink tube
migrate to Fired Torpedo
```

#### Migrating to Fired Implementation

```
<<active launch torpedo state model>>=
state Migrating_to_Fired() {
    // Include your C code for the state here.
}

final state Migrating_to_Fired
```

## Initial Instance Population

```
<<initial instances>>=
table
    Torpedo_Spec    (char const * Name)     (L_type Launch_type)
    @active         {"Active"}              {Active}
    @passive        {"Passive"}             {Passive}
end
```

```
<<initial instances>>=
table
    Storage_Rack    (unsigned Number)
    @sr1            {1}
    @sr2            {2}
    @sr3            {3}
    @sr4            {4}
    @sr5            {5}
    @sr6            {6}
    @sr7            {7}
    @sr8            {8}
end
```

```
<<initial instances>>=
table
    Tube            (Tube_number Number)
    @tb1            {1}
    @tb2            {2}
    @tb3            {3}
    @tb4            {4}
end
```

## Code Layout

### Root Chunk

```
<<launch.pycca>>=
domain launch
    <<interface prolog>>
    <<implementation prolog>>
    <<domain operations>>
    <<external operations>>
    <<classes>>
    <<initial instances>>
    <<interface epilog>>
    <<implementation epilog>>
end
```

### Class Chunks

For each of the classes in the domain, we define how the chunks are composed into the class specification. We have followed a naming convention that defines a chunk for attributes, references and state model for each class. Below we compose the components of the class definition into `pycca` syntax.

```
<<classes>>=
class Torpedo
    <<torpedo attributes>>
    <<torpedo references>>
    <<torpedo instance operations>>
    population dynamic
    slots 20
end

class Torpedo_Spec
    <<torpedo spec attributes>>

    population static
end

class Stored_Torpedo
    <<stored torpedo references>>
    machine
        <<stored torpedo state model>>
    end

    population dynamic
    slots 20
end

class Storage_Rack
    <<storage rack attributes>>
end

class Loaded_Torpedo
    <<loaded torpedo references>>

    population dynamic
    slots 8
end

class Tube
    <<tube attributes>>

    population static
end

class Fired_Torpedo
    <<fired torpedo references>>

    machine
        <<fired torpedo state model>>
    end

    population dynamic
    slots 20
end

class Passive_Launch_Torpedo
    <<passive launch torpedo references>>
    machine
        <<passive launch torpedo state model>>
    end

    population dynamic
    slots 8
end

class Active_Launch_Torpedo
    <<active launch torpedo references>>
    machine
```

```
        <<active launch torpedo state model>>
    end

    population dynamic
    slots 8
end
```

### Implemenation Prolog

```
<<implementation prolog>>=
implementation prolog {
    #include "launch.h"
    // Any additional implementation includes, etc.
    <<internal data types>>
}
```

### Interface Prolog

```
<<interface prolog>>=
interface prolog {
    #include <stdint.h>
    // Any additional interface includes, etc.
    <<external data types>>
}
```

## Literate Programming

The source for this document conforms to asciidoc syntax. This document is also a literate program. The source code for the implementation is included directly in the document source and the build process extracts the source that is then given to the Tcl interpreter. This process is known as *tangle*ing. The program, `atangle`, is available to extract source code from the document source and the `asciidoc` tool chain can be used to produce a variety of different output formats, although PDF is the intended choice.

The goal of a literate program is to explain the logic of the program in an order and fashion that facilitates human understanding of the program and then *tangle* the document source to obtain the Tcl code in an order suitable for the Tcl interpreter. Briefly, code is extracted from the literate source by defining a series of *chunks* that contain the source. A chunk is *defined* by including its name as:

```
<<chunk name>>=
```

The trailing = sign denotes a definition. A chunk definition ends at the end of the source block or at the beginning of another chunk definition. A chunk may be *referenced* from within a chunk definition by using its name without the trailing = sign, as in:

```
<<chunk definition>>=
    <<chunk reference>>
```

Chunk names are arbitrary strings. Multiple definitions with the same name are simply concatenated in the order they are encountered. There are one or more *root chunks* which form the conceptual tree for the source files that are contained in the literate source. By convention, root chunks are named the same as the file name to which they will be tangled. Tangling is then the operation of starting at a root chunk and recursively substituting the definition for the chunk references that are encountered.

For readers that are not familiar with the literate style and who are adept at reading source code directly, the chunks definitions and reordering provided by the tangle operation can be a bit disconcerting at first. You can, of course, examine the tangled source output, but if you read the program as a document, you will have to trust that the author managed to arrange the chunk definitions and references in a manner so that the tangled output is acceptable to further processing.