Air Traffic Controller Domain Workbook

A Pycca Translation

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
1.0	December 21, 2013	Initial release.	GAM
1.1	May 9, 2016	Updated state activities action language to match the latest syntax of the action language intended for the book.	GAM
1.2	February 3, 2017	Updated state activities action language to match the latest syntax of the action language intended for the book.	GAM
1.3	March 4, 2017	Minor edits to the external entity interfaces to match book text.	GAM

Contents

Introduction 1

Data Types 1

Employee_ID . . . 1
Station_Number . . . 1
Czone_Name . . . 2
Name_T . . . 2
Date_T . . . 2
Aircraft_Quantity . . . 2
Aircraft_Maximum . . . 2
Duration . . . 2
Experience_Level . . . 3

Classes and Relationships 3

Air Traffic Controller . . . 3
R1 Generalization . . . 4
Off Duty Controller . . . 4
On Duty Controller . . . 4
R3 — On Duty Controller ⇒ Duty Station . . . 5
Control Zone . . . 5
R2 — Control Zone ⇒ On Duty Controller . . . 5
Duty Station . . . 6
Shift Specification . . . 6

Active Classes 7

Air Traffic Controller Class State Model . . . 7
OFF_DUTY . . . 9
Verifying Adequate Break . . . 10
Logging In . . . 10
ON DUTY . . . 11
Handing off Control Zone . . . 11
Logging Out . . . 13
Verifying Full Handoff . . . 13
Duty Station Class State Model . . . 14
AVAILABLE . . . 16
IN USE . . . 16
MAX SHIFT EXCEEDED . . . 16
Canceling Shift Timeout . . . 17

Initial Instance population **17**

Air Traffic Controller Population . . . 17
On Duty Controller Population . . . 17
Control Zone Population . . . 18
Duty Station Population . . . 18
Shift Specification Population . . . 19

Domain Operations **19**

init . . . 19

External Operations **19**

Cannot handoff to self . . . 19
Unknown controller . . . 20
Zone not handled by controller . . . 20
Control Zones Active . . . 20
Break required . . . 21

Code Layout **21**

Class Chunks . . . 22
Prologues . . . 23

Literate Programming **23**

Index **25**

List of Figures

1 Air Traffic Control Center Class Diagram . . . 3
2 Air Traffic Controller / Duty Station Interaction . . . 7
3 Air Traffic Controller State Model Diagram . . . 8
4 Duty Station State Model Diagram . . . 15

List of Tables

1 Air Traffic Controller Transition Matrix . . . 9
2 Duty Station Transition Matrix . . . 15
3 Air Traffic Controller Population . . . 17
4 On Duty Controller Population . . . 17
5 Control Zone Population . . . 18
6 Duty Station Population . . . 18
7 Shift Specification Population . . . 19

Introduction

This document describes a domain for an Air Traffic Control application. The domain presented here is an example that is contained in the book, *Models To Code*. Needless to say, this domain does **not** represent the solution to a real Air Traffic Control problem and, by virtue of being an example, is necessarily limited in its functionality to better expose model translation concepts.

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 which 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 source using a literate programming tool named, `atangle`[2].

Important
This literate program document intermixes model elements and implementation elements. This clarifies the correspondence between the model and its translation and makes it easier to propagate model changes into the translation. However, we must emphasize 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 ATC domain defines a set of model level data types. A data type specifies a set of values that can be assigned to a model attribute, parameter, or temporary scalar variable. Each model level data type is described below and its implementation is given.

Note
The implementation of data types distinguishes between *external* and *internal data types*. External data types are those used in the domain and external operations and therefore must be included in the generated header file. Internal data types are used only within the domain and are not visible outside of the generated code file.

Employee_ID

The `Employee_ID` data type is the the set of all employee identifiers held as a character string.

Employee_ID Implementation

```
<<external data types>>=
typedef char const *Employee_ID ;
```

Station_Number

The `Station_Number` data type is the set of all control station names held as a character string.

Station_Number Implementation

```
<<external data types>>=
typedef char const *Station_Number ;
```

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

Czone_Name

The `Czone_Name` data type is the set of all Control Zone names held as a character string.

Czone_Name Implementation

```
<<external data types>>=
typedef char const *Czone_Name ;
```

Name_T

The `Name_T` data type is the set of all Air Traffic Controller names held as a character string.

Name_T Implementation

```
<<internal data types>>=
typedef char const *Name_T ;
```

Date_T

The `Date_T` data type is the set of dates used to record Air Traffic Controller activities.

Date_T Implementation There are many different implementation representations for date information. Here we use the standard POSIX time type.

```
<<internal data types>>=
typedef time_t Date_T ;
```

Aircraft_Quantity

The `Aircraft_Quantity` data type is the numeric type representing the number of aircraft that are in a Control Zone.

Aircraft_Quantity Implementation

```
<<internal data types>>=
typedef unsigned Aircraft_Quantity ;
```

Aircraft_Maximum

The `Aircraft_Maximum` data type is the set of values that describes the maximum number of aircraft that can be displayed on a Duty Station.

Aircraft_Maximum Implementation

```
<<internal data types>>=
typedef unsigned Aircraft_Maximum ;
```

Duration

The `Duration` data type is the set of values that represent a duration in time, measures in units of seconds.

Duration Implementation

```
<<internal data types>>=
typedef unsigned Duration ;
```

Experience_Level

The `Experience_Level` data type describes a rating system for Air Traffic Controllers indicating the amount of experience they have performing their job.

Experience_Level Implementation

```
<<internal data types>>=
typedef char const *Experience_Level ;
```

Classes and Relationships

The first facet of modeling a domain is to generate a class diagram. The figure below shows the class diagram fo the ATC domain.

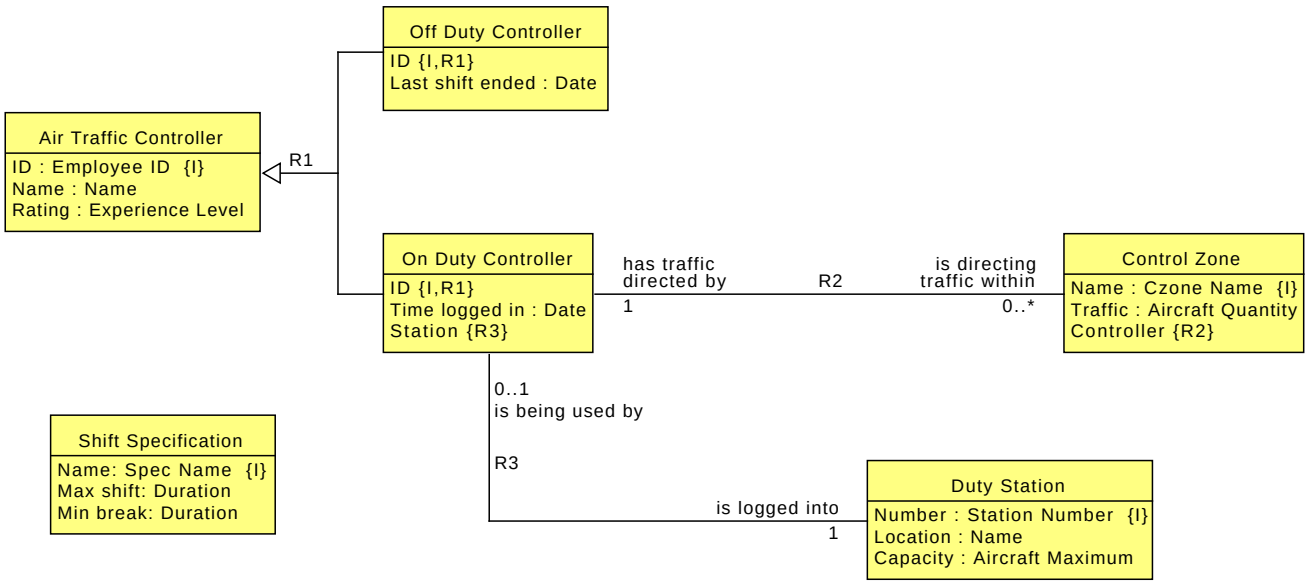

Figure 1: Air Traffic Control Center Class Diagram

Air Traffic Controller

The Air Traffic Controller class abstracts a human whose responsiblity is to monitor and guide the take off and landing of aircraft.

ID {I} : Employee_ID	An Air Traffic Controller is identified by a number assigned by the human resources department. This department is responsible for ensuring the uniqueness of the employee ID assigned to each Air Traffic Controller.
Name : Name	Each Air Traffic Controller also has a given name by which they are informally known. This name is not necessarily unique among the Air Traffic Controllers.
Rating : Experience_level	Air Traffic Controllers are also characterized by a rating which reflects the experience they have performing their job.

Air Traffic Controller Pycca Attributes

```
<<Air Traffic Controller attributes>>=
attribute (Employee_ID ID)
attribute (Name_T Name)
attribute (Experience_Level Rating)
```

R1 Generalization

This domain must ensure that officially mandated procedures are followed such as when and how an Air Traffic Controller is allowed to direct aircraft. The relevant data that is recorded and actions permitted revolve around controller work loads. Rules about initiating and handing off control center around the changing on and off duty roles of a controller.

The R1 generalization establishes the mutually exclusive roles of an Air Traffic Controller and provides a framework for managing a correct transition between these roles.

R1 Implementation

```
<<Air Traffic Controller references>>=
subtype R1 union
    Off_Duty_Controller
    On_Duty_Controller
end
```

To implement R1, we have chosen to store the instance data of the subclasses as a union composed into the data storage for the superclass. For simple generalizations such as this one, the choice of a union usually saves some memory space.

Off Duty Controller

An Off Duty Controller is an Air Traffic Controller who is not currently performing any control duties such as directing aircraft.

ID {I, R1}	An Off Duty Controller is identified in the same way as the related Air Traffic Controller.
Last shift ended : Date	The rules for ensuring adequate break time for an Air Traffic Controller require that we know when a controller's last duty shift was completed. This allows us to compute how long the controller has been on duty.

Off Duty Controller Pycca attributes

```
<<Off Duty Controller attributes>>=
attribute (Date_T Last_shift_ended) default {0}
```

On Duty Controller

An On Duty Controller is an Air Traffic Controller who is currently performing control duties.

ID {I, R1}	An On Duty Controller is identified in the same way as the related Air Traffic Controller.
Time logged in : Date	The time at which the On Duty Controller logged in.
Duty station {R3}	The On Duty Controller is currently logged into this Duty Station.

On Duty Controller Pycca attributes

```
<<On Duty Controller attributes>>=
attribute (Date_T Time_logged_in) default {0}
```

R3 — On Duty Controller ⇒ Duty Station

- **On Duty Controller** is logged into *exactly one* **Duty Station**
- **Duty Station** is being used by *zero or one* **On Duty Controller**

To perform control duties, an On Duty Controller must successfully log into an available Duty Station. Each Duty Station is either currently in use by a single controller or available for access. (For this exercise we will ignore the possibility of an out-of-service station).

R3 Implementation

```
<<Duty Station references>>=
reference R3 -> On_Duty_Controller

<<On Duty Controller references>>=
reference R3 -> Duty_Station
```

Control Zone

The airspace for flights is divided into distinct adjoining volumes called Control Zones. This partitioning of the space facilitates the management of aircraft by limiting the total quantity of aircraft that may occupy a given volume at the same time. It also provides a means for ensuring unambiguous responsibility for the handling of any given aircraft by a controller.

Name {I} : Czone name	Each Control Zone is identified by a name. The FAA is responsible for determining the names and ensuring that they are unique.
Traffic : Aircraft quantity	The number of aircraft currently traveling through the Control Zone's airspace.
Controller {R2}	The Employee ID of the controller currently managing aircraft in this Control Zone.

Control Zone Pycca attributes

```
<<Control Zone attributes>>=
attribute (Czone_Name Name)
attribute (Aircraft_Quantity Traffic)
```

R2 — Control Zone ⇒ On Duty Controller

- **Control Zone** has traffic directed by *exactly one* **On Duty Controller**
- **On Duty Controller** is directing traffic within *zero or more* **Control Zone**

A Control Zone may not ever be left unmanaged. To avoid potential confusion, the policy is to have only one controller responsible for a Control Zone at a time. It is also permissible for the same Air Traffic Controller to manage multiple Control Zones simultaneously. The individual controller may use their judgement (remember this is just an exercise!) to not assume an excessive workload.

In fact, a controller may be logged in, but not currently in charge of any particular Control Zone. This is typically during the period where the controller is starting and ending duty.

R2 Implementation

```
<<On Duty Controller references>>=
reference R2 –>>1 Control_Zone

<<Control Zone references>>=
reference R2 –> On_Duty_Controller
```

An analysis of the activities of the domain reveals that the relationship path from On Duty Controller to Control Zone is dynamic and must handle an arbitrary number of Control Zone references. So we have chosen a linked list type of reference for this path. The singular nature of the Control Zone to On Duty Controller path is handled by a simple singular reference.

Duty Station

A Duty Station is a work area designed for a single person consisting of a display, keyboard, and pointing device where air traffic can be directed in one or more Control Zones. Security and safety protocols require a log in process to provide access.

Number {I} : Station number	By policy, each station within an Air Traffic Control facility is labeled with a unique number. The number is issued when the station is first installed.
Location : Name	To make it easier to locate a given station, a general area is associated with it. This is an informal name relevant to the local facility.
Capacity : Aircraft maximum	The software configuration and physical design of a Duty Station imposes a maxiumum number of aircraft that can be safely managed at that station. This value is advisory and it is possible for the value to be exceeded temporarily. It is the active controller's responsibility to direct traffic such that this value is rarely, if ever, exceeded.

Duty Station Pycca attributes

```
<<Duty Station attributes>>=
attribute (Station_Number Number)
attribute (Name_T Location)
attribute (Aircraft_Maximum Capacity)
```

Shift Specification

The time durations and other potential future policy data that regulate the behavior of Air Traffic Controllers is established in this specification. Since these values apply to all controllers at all facilities, it is expected that there is only one instance in place. In the future, however, it may be the case that different policies may be in effect either at different times of the year (holiday vs. normal traffic) or possibly at different facilities. In that case, multiple instances may be maintained, so this abstraction also serves as a placeholder for future exansion.

Name {I} : Name	A unique and descriptive name. In the current system where there is only one instance, it may as well be "the spec". In the future, it may be something like "Holiday traffic rules" or "Normal traffic rules".
Min break : Duration	When an Air Traffic Controller goes off duty, they are required to remain off duty for at least this continuous period of time. It will not be possible for this controller to log in again to a Duty Station until this time has elapsed.

Max shift : Duration	This is the maximum continuous interval that an Air Traffic Controller may be logged in before they are required to take a mandated break. When a controller logs off before this duration has elapsed (for this exercise anyway) a full break is required.

Shift Specification Pycca attributes

```
<<Shift Specification attributes>>=
attribute (Duration Min_break)
attribute (Duration Max_shift)
```

Active Classes

Here is the collaboration and sequencing among the classes having modeled lifeycles.

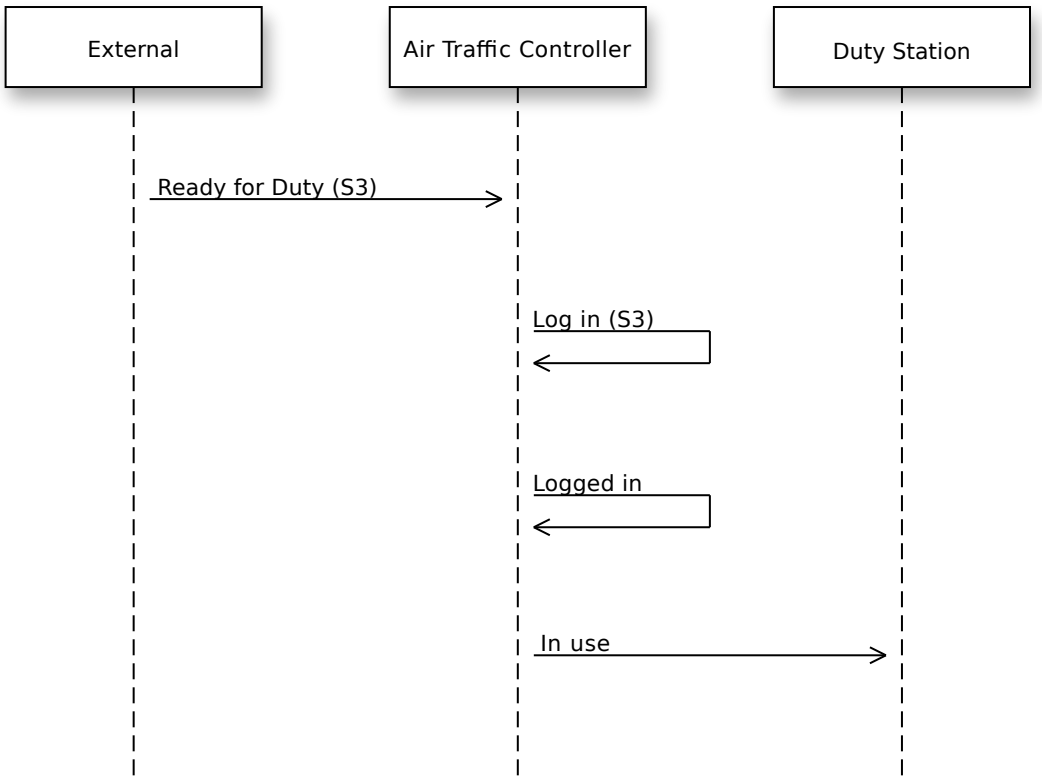

Figure 2: Air Traffic Controller / Duty Station Interaction

Air Traffic Controller Class State Model

The Air Traffic Controller state model manages the migration of an ATC between on and off duty roles, ensuring availabity of a necessary Duty Station and proper handoff of all Control Zones before going off duty.

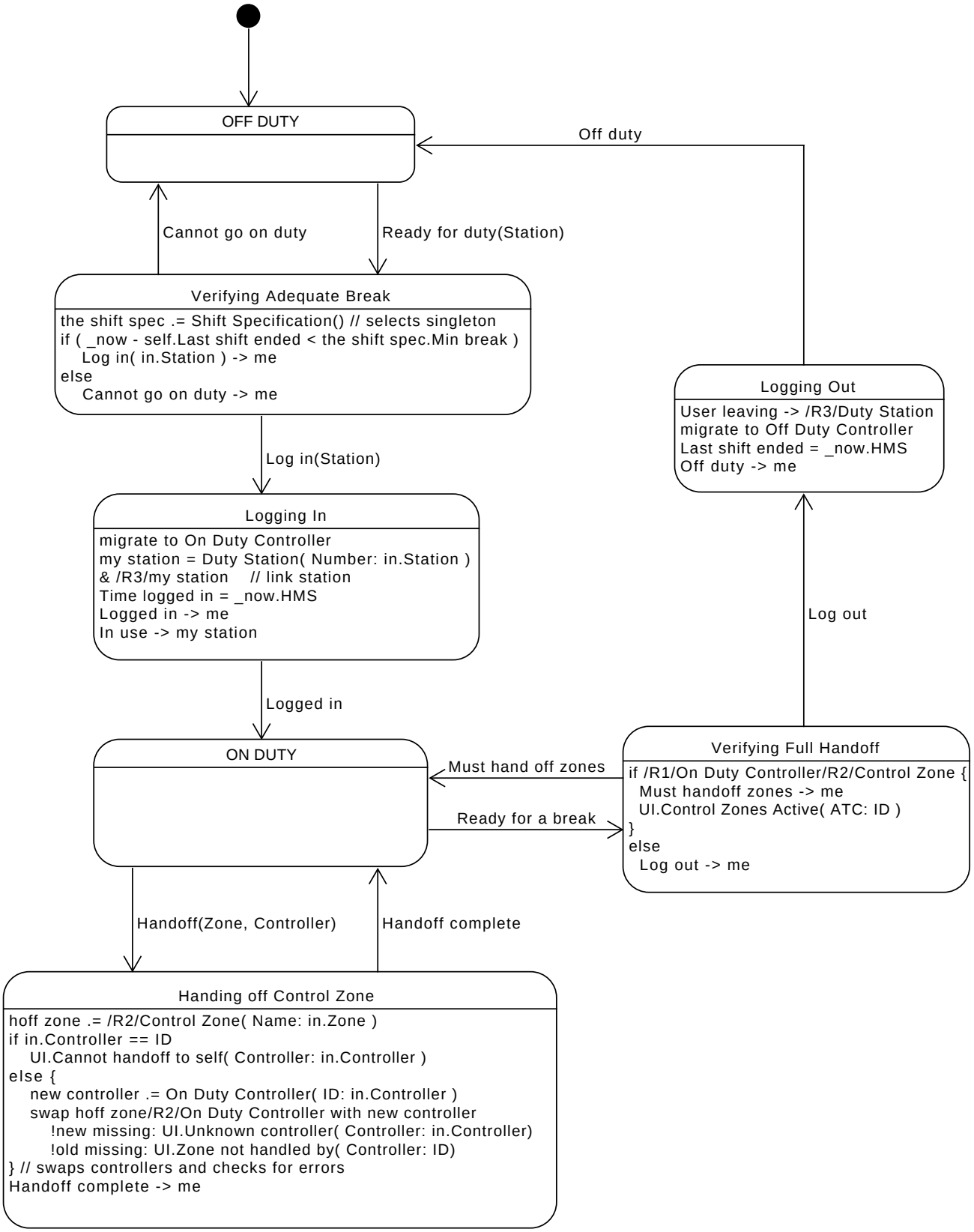

Figure 3: Air Traffic Controller State Model Diagram

Table 1: Air Traffic Controller Transition Matrix

	Ready for duty	**Log in**	**Cannot go on duty**	**Logged in**	**Ready for a break**	**Handoff**	**Handoff complete**	**Off duty**	**Log out**
OFF DUTY	Verifying Adequate Break	CH	CH	CH	CH	CH	CH	CH	CH
Verifying Adequate Break	CH	Logging In	OFF DUTY	CH	CH	CH	CH	CH	CH
Logging In	CH	CH	CH	ON DUTY	CH	CH	CH	CH	CH
ON DUTY	CH	CH	CH	CH	Verifying Full Handoff	Handing off Control Zone	CH	CH	CH
Handing off Control Zone	CH	CH	CH	CH	CH	CH	On Duty	CH	CH
Logging Out	CH	CH	CH	CH	CH	CH	CH	OFF DUTY	CH
Verifying Full Handoff	CH	CH	CH	CH	CH	CH	CH	CH	Logging Out

Air Traffic Controller Pycca Transitions

```
<<Air Traffic Controller state model>>=
default transition CH
initial state OFF_DUTY

transition OFF_DUTY - Ready_for_duty -> Verifying_Adequate_Break

transition Verifying_Adequate_Break - Log_in -> Logging_In
transition Verifying_Adequate_Break - Cannot_go_on_duty -> OFF_DUTY

transition Logging_In - Logged_in -> ON_DUTY

transition ON_DUTY - Ready_for_a_break -> Verifying_Full_Handoff
transition ON_DUTY - Handoff -> Handing_off_Control_Zone

transition Handing_off_Control_Zone - Handoff_complete -> ON_DUTY

transition Logging_Out - Off_duty -> OFF_DUTY

transition Verifying_Full_Handoff - Log_out -> Logging_Out
transition Verifying_Full_Handoff - Must_hand_off_zones -> ON_DUTY
```

OFF_DUTY

The ATC is not logged in to a Duty Station or otherwise directing air traffic.

Activity

```
// no activity for this state
```

Implementation

```
<<Air Traffic Controller state model>>=
state OFF_DUTY ()
{
    // no activity for this state
}
```

Verifying Adequate Break

Transitory state: An ATC is trying to go on duty. Here the Shift Specification is consulted to see if the ATC has spent an adequate interval on break. If not, they must remain Off Duty.

Activity

```
the shift spec .= Shift Specification() // selects singleton
if ( _now - self.Last shift ended < the shift spec.Min break )
    Log in( in.Station ) -> me
else
    Cannot go on duty -> me
```

Implementation

```
<<Air Traffic Controller state model>>=
state Verifying_Adequate_Break(Station_Number Station)
{
    //+ the shift spec .= Shift Specification() // selects singleton
    ClassRefVar(Shift_Specification, the_shift_spec) =
            Instance(Shift_Specification, singleton) ;

    ClassRefVar(Off_Duty_Controller, offdc) =
        PYCCA_unionSubtype(self, R1, Off_Duty_Controller) ;

    //+ if ( _now - self.Last shift ended < the shift spec.Min break )
    if (offdc->Last_shift_ended == 0 ||
            time(NULL) - offdc->Last_shift_ended < the_shift_spec->Min_break) {
        //+ Log in( in.Station ) -> me
        MechEcb li_sig = PYCCA_newEvent(Log_in, Air_Traffic_Controller, self,
                self) ;
        PYCCA_eventParam(li_sig, Air_Traffic_Controller, Log_in, Station) =
                rcvd_evt->Station ;
        PYCCA_postSelfEvent(li_sig) ;
    } else {
        //+ Cannot go on duty -> me
        PYCCA_generateToSelf(Cannot_go_on_duty) ;
    }
}
```

Logging In

The Off Duty Controller migrates to On Duty and logs into the requested Duty Station. LS: I see that we are not verifying that the Station is available. This is an error that should be dealt with here. Exercise for the student?

Activity

```
migrate to On Duty Controller
my station = Duty Station( Number: in.Station )
& /R3/my station    // link station
Time logged in = _now.HMS
Logged in -> me
In use -> my station
```

Implementation

```
<<Air Traffic Controller state model>>=
state Logging_In(Station_Number Station)
{
    //+ migrate to On Duty Controller
    PYCCA_migrateSubtype(self, Air_Traffic_Controller, R1,
            On_Duty_Controller) ;

    //+ my station = Duty Station( Number: in.Station )
    ClassRefVar(Duty_Station, my_station) ;
    PYCCA_selectOneStaticInstWhere(my_station, Duty_Station,
            strcmp(my_station->Number, rcvd_evt->Station) == 0) ;
    assert(my_station != EndStorage(Duty_Station)) ;

    //+ & /R3/my station    // link station
    ClassRefVar(On_Duty_Controller, ondc) =
            PYCCA_unionSubtype(self, R1, On_Duty_Controller) ;
    ondc->R3 = my_station ;
    my_station->R3 = ondc ;

    //+ Time logged in = _now.HMS
    ondc->Time_logged_in = time(NULL) ;

    //+ Logged in -> me
    PYCCA_generateToSelf(Logged_in) ;

    //+ In use -> my station
    PYCCA_generate(In_use, Duty_Station, my_station, self) ;
}
```

ON DUTY

The ATC has logged in successfully and may now acquire one or more Control Zones and direct air traffic.

Activity

```
// no activity for this state
```

Implementation

```
<<Air Traffic Controller state model>>=
state ON_DUTY()
{
    // no activity for this state
}
```

Handing off Control Zone

Transitory state: Before going off duty, an ATC must hand-off each of their Control Zones. In this state an attempt is made to hand-off a single Control Zone to some other controller.

Activity

```
hoff zone .= /R2/Control Zone( Name: in.Zone )
if in.Controller == ID
    UI.Cannot handoff to self( Controller: in.Controller )
else {
    new controller .= On Duty Controller( ID: in.Controller )
    swap hoff zone/R2/On Duty Controller with new controller
        !new missing: UI.Unknown controller( Controller: in.Controller)
        !old missing: UI.Zone not handled by( Controller: ID)
} // swaps controllers and checks for errors
Handoff complete -> me
```

Implementation

```
<<Air Traffic Controller state model>>=
state Handing_off_Control_Zone(
    Czone_Name zone,
    Employee_ID controller)
{
    //+ hoff zone .= /R2/Control Zone( Name: in.Zone )
    ClassRefVar(On_Duty_Controller, ondc) =
            PYCCA_unionSubtype(self, R1, On_Duty_Controller) ;
    ClassRefVar(Control_Zone, hoff_zone) = NULL ;
    rlink_t *czlink ;
    PYCCA_forAllLinkedInst(ondc, R2, czlink) {
        ClassRefVar(Control_Zone, found) =
                PYCCA_linkToInstRef(czlink, Control_Zone, R2) ;
        if (strcmp(rcvd_evt->zone, found->Name) == 0) {
            hoff_zone = found ;
            break ;
        }
    }
    assert(hoff_zone != NULL) ;

    //+ if in.Controller == ID
    if (strcmp(rcvd_evt->controller, self->ID) == 0) {
        //+ UI.Cannot handoff to self( Controller: in.Controller )
        ExternalOp(Cannot_handoff_to_self)(rcvd_evt->controller) ;
    } else {
        //+ new controller .= On Duty Controller( ID: in.Controller )
        ClassRefVar(Air_Traffic_Controller, new_controller) ;
        PYCCA_selectOneStaticInstWhere(new_controller,
            Air_Traffic_Controller,
            strcmp(new_controller->ID, rcvd_evt->controller) == 0 &&
            new_controller->SubCodeMember(R1) ==
                SubCodeValue(Air_Traffic_Controller, R1, On_Duty_Controller)) ;

        if (new_controller == EndStorage(Air_Traffic_Controller)) {
            //+ !new missing: UI.Unknown controller( Controller: in.Controller)
            ExternalOp(Unknown_controller)(rcvd_evt->controller) ;
        } else if (hoff_zone->R2 != ondc) {
            //+ !old missing: UI.Zone not handled by( Controller: ID)
            ExternalOp(Zone_not_handled_by_controller)(self->ID) ;
        } else {
            //+ swap hoff zone/R2/On Duty Controller with new controller
            ClassRefVar(On_Duty_Controller, new_ondc) =
                PYCCA_unionSubtype(new_controller, R1,
                    On_Duty_Controller) ;
            PYCCA_unlinkFromMany(hoff_zone, R2) ;
            PYCCA_linkToMany(new_ondc, R2, hoff_zone) ;
            hoff_zone->R2 = new_ondc ;
        }
```

```
    }

    //+ Handoff complete -> me
    PYCCA_generateToSelf(Handoff_complete) ;
}
```

Logging Out

Transitory state: At this point an ATC wanting to go off duty is not managing any Control Zones. Now the ATC logs out and migrates to an Off Duty Controller.

Activity

```
User leaving -> /R3/Duty Station
migrate to Off Duty Controller
Last shift ended = _now.HMS
Off duty -> me
```

Implementation

```
<<Air Traffic Controller state model>>=
state Logging_Out()
{
    //+ User leaving -> /R3/Duty Station
    assert(self->SubCodeMember(R1) ==
            SubCodeValue(Air_Traffic_Controller, R1,
            On_Duty_Controller)) ;
    ClassRefVar(On_Duty_Controller, ondc) =
            PYCCA_unionSubtype(self, R1, On_Duty_Controller) ;
    ClassRefVar(Duty_Station, ds) = ondc->R3 ;
    assert(ds != NULL) ;
    PYCCA_generate(User_leaving, Duty_Station, ds, self) ;

    //+ migrate to Off Duty Controller
    ondc->R3 = NULL ;
    ds->R3 = NULL ;
    PYCCA_migrateSubtype(self, Air_Traffic_Controller, R1,
            Off_Duty_Controller) ;

    //+ Last shift ended = _now.HMS
    ClassRefVar(Off_Duty_Controller, offdc) =
            PYCCA_unionSubtype(self, R1, Off_Duty_Controller) ;
    offdc->Last_shift_ended = time(NULL) ;

    //+ Off duty -> me
    PYCCA_generateToSelf(Off_duty) ;
}
```

Verifying Full Handoff

Transitory state: The ATC has signaled a desire to go Off Duty, but here we verify that this ATC is no longer managing any Control Zones. Otherwise, they must remain on duty.

Activity

```
if /R1/On Duty Controller/R2/Control Zone {
  Must handoff zones -> me
  UI.Control Zones Active( ATC: ID )
}
else
  Log out -> me
```

Implementation

```
<<Air Traffic Controller state model>>=
state Verifying_Full_Handoff()
{
    ClassRefVar(On_Duty_Controller, ondc) =
            PYCCA_unionSubtype(self, R1, On_Duty_Controller) ;

    //+ if /R1/On Duty Controller/R2/Control Zone {
    if (PYCCA_isLinkNotEmpty(ondc, R2)) {
        //+ Must handoff zones -> me
        PYCCA_generateToSelf(Must_hand_off_zones) ;

        //+  UI.Control Zones Active( ATC: ID )
        ExternalOp(Control_Zones_Active)(self->ID) ;
    } else {
        //+ Log out -> me
        PYCCA_generateToSelf(Log_out) ;
    }
}
```

Duty Station Class State Model

A Duty Station is AVAILABLE when no ATC is logged in. (For this exercise we won't worry about out of order stations). Eventually an ATC logs in, performs their shift and logs out again leaving the station AVAILABLE again. The Duty Station will trigger a warning if the shift exceeds the maximum shift duration.

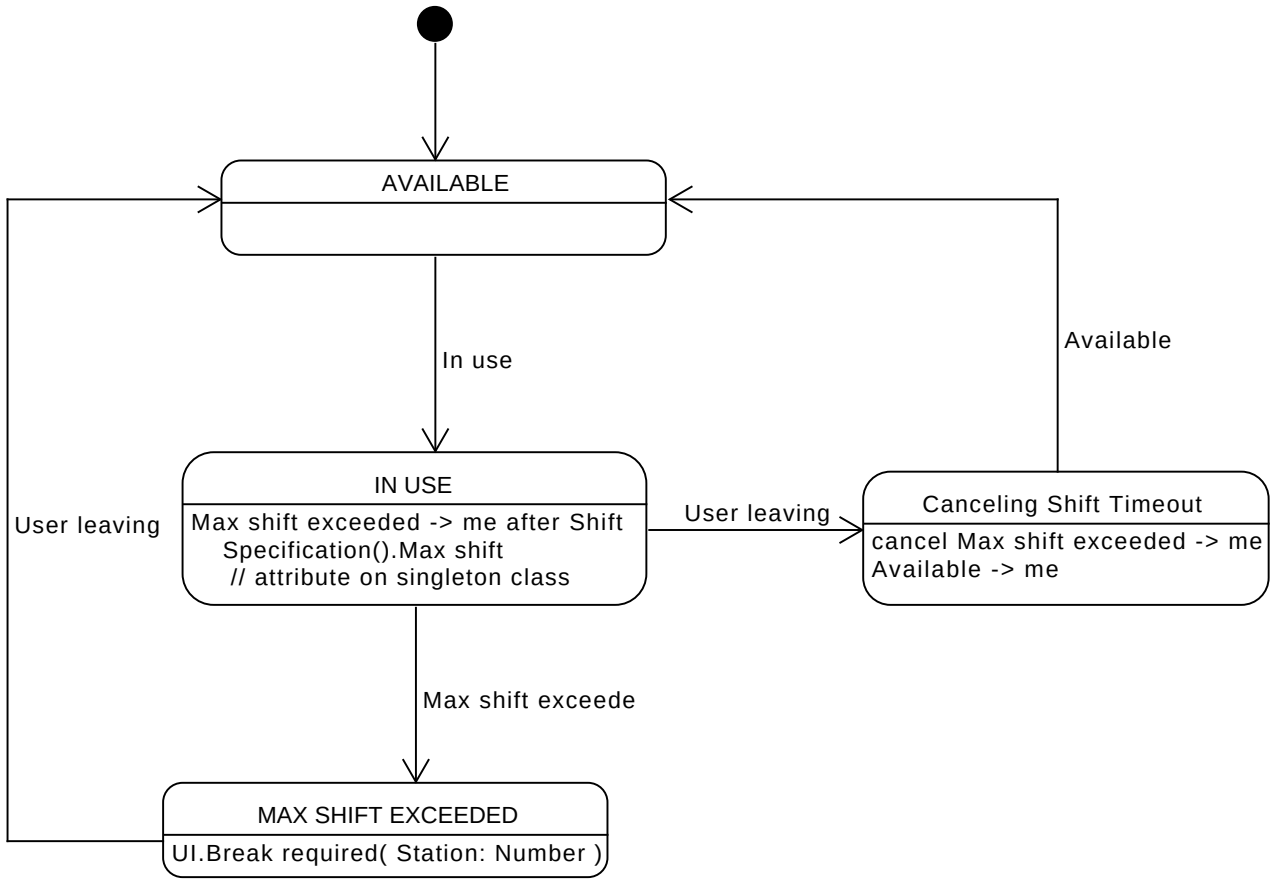

Figure 4: Duty Station State Model Diagram

Table 2: Duty Station Transition Matrix

	In use	Max shift exceeded	User leaving	Available
AVAILABLE	IN USE	CH	CH	CH
IN USE	CH	MAX SHIFT EXCEEDED	Canceling Shift Timeout	CH
MAX SHIFT EXCEEDED	CH	CH	AVAILABLE	CH
Canceling Shift Timeout	CH	CH	CH	AVAILABLE

Duty Station Pycca Transitions

```
<<Duty Station state model>>=
default transition CH
initial state AVAILABLE

transition AVAILABLE - In_use -> IN_USE

transition IN_USE - Max_shift_exceeded -> MAX_SHIFT_EXCEEDED
transition IN_USE - User_leaving -> Canceling_Shift_Timeout
```

```
transition MAX_SHIFT_EXCEEDED - User_leaving -> AVAILABLE

transition Canceling_Shift_Timeout - Available -> AVAILABLE
```

AVAILABLE

No ATC is logged into the Duty Station at the moment. It is operational and available for use.

Activity

```
// no activity in this state
```

Implementation

```
<<Duty Station state model>>=
state AVAILABLE()
{
    // no activity in this state
}
```

IN USE

Activity

```
Max shift exceeded -> me after Shift
    Specification().Max shift
     // selects singleton
```

Implementation

```
<<Duty Station state model>>=
state IN_USE()
{
    //+ selects singleton
    ClassRefVar(Shift_Specification, ss) =
            Instance(Shift_Specification, singleton) ;

    //+ Max shift exceeded -> me after Shift Specification().Max shift
    PYCCA_generateDelayedToSelf(Max_shift_exceeded,
            ss->Max_shift * 1000UL) ; // delay times are in ms
}
```

MAX SHIFT EXCEEDED

The ATC has exceeded their allowed shift duration. A warning is flagged.

Activity

```
UI.Break required( Station: Number )
```

Implementation

```
<<Duty Station state model>>=
state MAX_SHIFT_EXCEEDED()
{
    //+UI.Break required( Station: Number )
    ExternalOp(Break_required)(self->Number) ;
}
```

Canceling Shift Timeout

Transitory state: The Max Shfit Exceeded delayed event is canceled and the station becomes AVAILABLE again.

Activity

```
cancel Max shift exceeded -> me
Available -> me
```

Implementation

```
<<Duty Station state model>>=
state Canceling_Shift_Timeout()
{
    //+ cancel Max shift exceeded -> me
    PYCCA_cancelDelayedToSelf(Max_shift_exceeded) ;

    //+ Available -> me
    PYCCA_generateToSelf(Available) ;
}
```

Initial Instance population

This population matches the scenario used in Models to Code.

Air Traffic Controller Population

Table 3: Air Traffic Controller Population

ID{I}	Name	Rating
53	Toshiko	A
67	Gwen	B
51	Ianto	C

Implementation

```
<<population>>=
table
Air_Traffic_Controller (Employee_ID ID) (Name_T Name)
        (Experience_Level Rating) R1
@atc53  {"53"} {"Toshiko"}  {"A"} -> On_Duty_Controller.atc53
@atc67  {"67"} {"Gwen"}     {"B"} -> On_Duty_Controller.atc67
@atc51  {"51"} {"Ianto"}    {"C"} -> On_Duty_Controller.atc51
end
```

On Duty Controller Population

Table 4: On Duty Controller Population

ID{I, R1}	Time logged in	Duty Station
53	Now	S2

Table 4: (continued)

ID{I, R1}	Time logged in	Duty Station
67	Now	S1
51	Now	S3

Implementation

```
<<population>>=
table
On_Duty_Controller          R2              R3
@atc53                      ->> sfo end     -> s2
@atc67                      ->> oak end     -> s1
@atc51                      ->> sjc end     -> s3
end
```

Control Zone Population

Table 5: Control Zone Population

Name{I, R1}	Traffic	Controller {R2}
SFO37B	27	53
OAK21C	18	67
SJC18C	9	51

Implementation

```
<<population>>=
table
Control_Zone    (Czone_Name Name)   (Aircraft_Quantity Traffic) R2
@sfo            {"SFO37B"}          {27}                        -> atc53
@oak            {"OAK21C"}          {18}                        -> atc67
@sjc            {"SJC18C"}          {9}                         -> atc51
end
```

Duty Station Population

Table 6: Duty Station Population

Number{I}	Location	Capacity
S1	Front	20
S2	Center	30
S3	Front	45

Implementation

```
<<population>>=
table
Duty_Station    (Station_Number Number) (Name_T Location) (Aircraft_Maximum Capacity)
@s1             {"S1"}                  {"Front"}         {20}
@s2             {"S2"}                  {"Center"}        {30}
@s3             {"S3"}                  {"Front"}         {45}
end
```

Shift Specification Population

Table 7: Shift Specification Population

Name{I}	Min break	Max shift
singleton	15 min	2 hr 15 min

Implementation

```
<<population>>=
instance Shift_Specification@singleton
    (Duration Min_break)    {15 * 60}               # 15 minutes in seconds
    (Duration Max_shift)    {(2 * 60 + 15) * 60}    # 2 hr 15 min in seconds
end
```

Domain Operations

init

An `init()` operation is provided and must be invoked at domain initialization time.

init Implementation

```
<<domain operations>>=
domain operation
init()
{
    // Your code here.
}
```

External Operations

Cannot handoff to self

```
Cannot_handoff_to_self(controller: Employee_ID)
```

`controller`
The identifier of the controller which attempt to handoff a control zone to himself.

Implementation

```
<<external operations>>=
external operation
Cannot_handoff_to_self(
    Employee_ID controller)
{
    printf("Cannot Handoff to Self: \"%s\"\n", controller) ;
}
```

Unknown controller

```
Unknown_controller(controller: Employee_ID)
```

`controller`
The identifier of the controller which was not found during a control zone handoff.

Implementation

```
<<external operations>>=
external operation
Unknown_controller(
    Employee_ID controller)
{
    printf("Bad Handoff Target: \"%s\"\n", controller) ;
}
```

Zone not handled by controller

```
Zone_not_handled_by_controller(controller: Employee_ID)
```

`controller`
The identifier of the controller which does not control a zone.

Implementation

```
<<external operations>>=
external operation
Zone_not_handled_by_controller(
    Employee_ID controller)
{
    printf("Control zone not handled by \"%s\"\n", controller) ;
}
```

Control Zones Active

```
Control Zones Active( ATC: Employee ID )
```

`ATC`
The identifier of the controller which does not control a zone.

Implementation

```
<<external operations>>=
external operation
Control_Zones_Active(
    Employee_ID atc)
{
    printf("Control Zones Active: %s\n", atc) ;
}
```

Break required

```
Break required ( Station: Station Number )
```

`Station`
The identifier of the Duty Station whose controller requires a break.

Implementation

```
<<external operations>>=
external operation
Break_required(
    Station_Number station)
{
    printf("Break required: %s\n", station) ;
}
```

Code Layout

The order of components in a `pycca` file is somewhat arbitrary. The only order imposed by `pycca` itself is that class definitions must precede class populations. The generated "C" file is reordered by `pycca` to meet the needs of the compiler. Generally this means that definitions appear before their use and thus inverts the more natural order of code presentation. This is only significant because it is sometimes the "C" file that is viewed in a debugger.

Root Chunk

```
<<atctrl.pycca>>=
# DO NOT EDIT THIS FILE!
# THIS FILE IS GENERATED FROM THE SOURCE OF A LITERATE PROGRAM.
# YOU MUST EDIT THE ORIGINAL SOURCE TO MODIFY THIS FILE.
#*++
# Copyright 2017 by Leon Starr, Andrew Mangogna and Stephen Mellor
#
# 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.
#
# Project:
#   Models to Code Book
#
# Module:
#   Air Traffic Controller Domain pycca file
#*--

domain atctrl
    <<interface prolog>>
    <<interface epilog>>
    <<domain operations>>
    <<external operations>>
    <<classes>>
    <<population>>
    <<implementation prolog>>
    <<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>>=
<<Air Traffic Controller class>>
<<Off Duty Controller class>>
<<On Duty Controller class>>
<<Control Zone class>>
<<Duty Station class>>
<<Shift Specification class>>
```

Air Traffic Controller Class

```
<<Air Traffic Controller class>>=
class Air_Traffic_Controller
    <<Air Traffic Controller attributes>>
    <<Air Traffic Controller references>>
    machine
        <<Air Traffic Controller state model>>
    end
end
```

Off Duty Controller Class

```
<<Off Duty Controller class>>=
class Off_Duty_Controller
    <<Off Duty Controller attributes>>
end
```

On Duty Controller Class

```
<<On Duty Controller class>>=
class On_Duty_Controller
    <<On Duty Controller attributes>>
    <<On Duty Controller references>>
end
```

Control Zone Class

```
<<Control Zone class>>=
class Control_Zone
    <<Control Zone attributes>>
    <<Control Zone references>>
end
```

Duty Station Class

```
<<Duty Station class>>=
class Duty_Station
    <<Duty Station attributes>>
    <<Duty Station references>>
    machine
        <<Duty Station state model>>
    end
end
```

Shift Specification Class

```
<<Shift Specification class>>=
class Shift_Specification
    <<Shift Specification attributes>>
end
```

Prologues

Implemenation Prolog

```
<<implementation prolog>>=
implementation prolog {
    // Any additional implementation includes, etc.
    #include <assert.h>
    #include <time.h>
    #include <string.h>
    #include "atctrl.h"
    <<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. 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 unfamiliar 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.

Index

A
Aircraft_Maximum, 2
Aircraft_Quantity, 2

C
Czone_Name, 2

D
data types
    Aircraft_Maximum, 2
    Aircraft_Quantity, 2
    Czone_Name, 2
    Date_T, 2
    Duration, 2
    Employee_ID, 1
    Experience_Level, 3
    Name_T, 2
    Station_Number, 1
Date_T, 2
Duration, 2

E
Employee_ID, 1
Experience_Level, 3

N
Name_T, 2

S
Station_Number, 1