Lubrication Domain Workbook

A Pycca Translation

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME
0.1	April 15, 2016	Initial coding.	GAM
1.0	June 8, 2016	Clean up and rework prior to Chapter 6 writing.	GAM
1.1	Sept 3, 2016	Incorporating changes and clean up after first draft of Chapter 6.	GAM
1.2	March 5, 2017	Clean up for release to book site.	GAM

Contents

List of Figures

List of Tables

Introduction

This document describes the Lubrication domain. This domain is part of the Automatic Lubrication System. 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 Automatic Lubrication problem and, by virtue of being an example, is necessarily limited in it functionality to expose better model translation concepts.

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

Each domain defines a set of model level data types. Model data types specify a set of values that model attributes may possess.

Count
Attributes of `Count` type are used to count things as conventional non-negative integers.

```
<<external data types>>=
typedef uint32_t Count ;
```

Seconds
A time value consisting of an integral number of seconds.

```
<<internal data types>>=
typedef uint32_t Seconds ;
```

Duration
A time value ???.

```
<<internal data types>>=
typedef uint32_t Duration ;
```

MPa
A pressure value consisting of an integral number of megapascals.

```
<<internal data types>>=
typedef uint32_t MPa ;
```

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

Name_t

A string value used to give arbitrary names to things. Note the naming difference to the class diagram. `Pycca` has a limitation that prevents type names begin the same as attribute names.

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

Model Name

A string value containing the name of injector models.

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

Fluid_State

The set of values that represents the amount of lubricating fluid in a reservoir. The amounts are represented in relative amounts rather than a physical measure of the lubrication volume.

```
<<internal data types>>=
typedef enum {
    FS_normal,
    FS_low,
    FS_verylow,
    FS_empty
} Fluid_State ;
```

Classes and Relationships

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

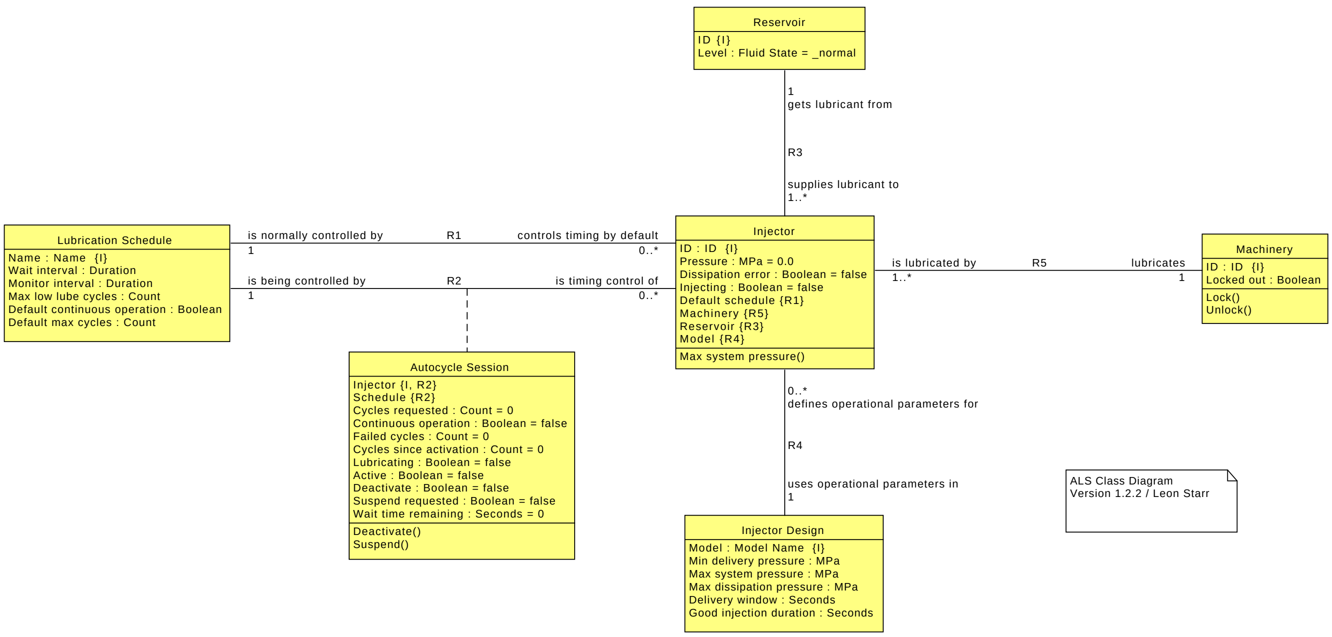

Figure 1: Automatic Lubrication System Class Diagram

Lubrication Schedule

Defines the key time periods and repetition behavior of an Injector.

The intervals vary depending on what kind of lubricant is delivered, what type of machinery is being lubricated and what sorts of operational modes are required.

Name {I}	This name can reflect the intended mode of operation such as “Manual”, “Test”, “Low Temperature” or “High Activity”. It may also refer to the type of lubricant used as this may affect what type of cycle should be applied such as “Grease” or “Oil 10-30”. **Data Type:** Name
Wait interval	This is the amount of time to wait in between lubrications. **Data Type:** Duration
Monitor interval	The lubricant pressure must be monitored prior to injection to insure that the pressure from the previous injection has adequately dissipated. This is the period prior to injection where the monitoring begins. **Data Type:** Duration
Max low lube cycles	When the lubricant runs low, the lubrication cycle will stop and an error will be issued. Each time low lubricant is detected, a count is maintained. The count is reset if the low level state goes back to normal. But if the count reaches the maximum specified here, automatic lubrication will stop. **Data Type:** Count
Default continuous operation	If set, the Default max cycles attribute is ignored and each time the Autocycle Session completes, it will repeat automatically. **Data Type:** Boolean
Default max cycles	If continuous operation is not set, this is the number of times the Lubrication Cycle will repeat until it shuts itself off automatically. Otherwise, this value is ignored. **Data Type:** Count

Lubrication Schedule Spec Pycca attributes

```
<<Lubrication Schedule attributes>>=
attribute (Name_t Name)
attribute (Duration Wait_interval)
attribute (Duration Monitor_interval)
attribute (Count Max_low_lube_cycles)
attribute (bool Default_continuous_operation)
attribute (Count Default_max_cycles)
```

Injector Design

A variety of parameters govern the safe and reliable control of injections depending on the particular design of an injector. All of these are captured in this specification.

Model {I}	Each Injector Design is identified by the injector model name. **Data Type:** Model Name
Min delivery pressure	The lubrication system should be above this pressure in order for injection to be effective. This is typically 15 MPa. If an injection cannot stay above this pressure during the Good inject duration, it is considered a low pressure injection. **Data Type:** MPa

Max system pressure	The maximum safe pressure in the lubrication system. A warning should be issued and the system shut down if this pressure is exceeded for too long (see below). Typically 26 MPa. **Data Type:** MPa
Max dissipation pressure	In between injections, pressure should dissipate. Before starting a new injection, this value is checked to ensure that the pressure is not any higher. Typically 26 MPa. **Data Type:** MPa
Delivery window	This is the maximum amount of time it takes to perform a complete injection. Typically 90 to 180 seconds. **Data Type:** Seconds
Good injection duration	The amount of time an injection with adequate delivery pressure should take. This is typically 7-12 seconds. Delivery is stopped if the pressure drops below the delivery threshold and must be restarted all over again when pressure is available. **Data Type:** Seconds

Injector Design Pycca attributes

```
<<Injector Design attributes>>=
attribute (Model_Name Model)
attribute (MPa Min_delivery_pressure)
attribute (MPa Max_system_pressure)
attribute (MPa Max_dissipation_pressure)
attribute (Seconds Delivery_window)
attribute (Seconds Good_injection_duration)
```

Injector

An injection system delivers lubricant from a reservoir, through a network of supply lines out through one or more injection points. This system is commonly referred to more simply as an injector. Injection is typically triggered by energizing a solenoid. It is important not to confuse the Injector with an individual injection point.

ID {I}	This is an arbitrary value used for identification purposes only. By policy, each Machinery unit has a unique arbitrary identifier. **Data Type:** ID
Pressure	The current detected pressure in the supply lines. **Data Type:** MPa
Dissipation error	If the pressure in the lines has not dissipated adequately after the previous injection a ‘Dissipation Error’ alarm will be raised. But it should only be raised once during a cycle. This setting remembers whether or not the error was previously raised in the current cycle. **Data Type:** Boolean
Injecting	The Good injection time remaining attribute is relevant only while an injection is in progress. Any value will be meaningless unless Injecting has been set. **Data Type:** Boolean
Default schedule	Each Injector has a schedule that normally controls it.
Machinery {R5}	The Machinery to be lubricated.
Reservoir {R3}	The lubricant reservoir attached to the injector.

Model {R4} The design model of the Injector.

Injector Pycca attributes

```
<<Injector attributes>>=
attribute (MPa Pressure) default {0}
attribute (bool Dissipation_error) default {false}
attribute (bool Injecting) default {false}
```

R4 — Injector ⇒ Injector Design

- **Injector** uses operational parameters in *exactly one* **Injector Design**
- **Injector Design** defines operational parameters for *zero or more* **Injector**

An Injector Design represents the design parameters that apply to all Injectors built from the same design. Since Injectors may be changed in the field, it can be desirable to keep Injector Designs installed that do not currently correspond to any installed Injectors.

R4 Implementation

```
<<Injector references>>=
reference R4 -> Injector_Design
```

R5 — Injector ⇒ Machinery

- **Machinery** is lubricated by *one or more* **Injector**
- **Injector** lubricates *exactly one* **Machinery**

An assembly of equipment serviced by automatic lubrication can be organized into one or more units of Machinery. There are two considerations that determine the best organization.

1. Are multiple different but simultaneous Autocycle Sessions required? In other words, do different parts of the equipment have different lubrication requirements?
2. Can all physical locations on a unit of Machinery be physically accessed by the same Injector system?

An Injector provides lubrication to one or more points on a single unit of Machinery.

R5 Implementation

```
<<Injector references>>=
reference R5 -> Machinery

<<Machinery references>>=
reference R5 ->>c Injector
```

Autocycle Session

Each Injector is driven by a fixed sequence of time intervals that repeat cyclically. The durations of these intervals are determined by an associated Cycle Spec. The Autocycle Session regulates the timing for an individual Injector.

Injector {R2}	The lubricating Injector controlled by this Autocycle Session.
Schedule {R2}	The Lubrication Schedule used by this Autocycle Session.
Cycles requested	This is the number of default cycles and, hence, injections that will be commanded. This value is ignored if Continuous operation is set. **Data Type:** Count
Continuous operation	If set, the cycle will keep repeating until it is stopped by an operator or some environmental conditions such as low lube level. The Cycles requested attribute is ignored when Continuous operation is set. **Data Type:** Boolean
Failed cycles	When lubrication does not succeed, due to a low level of lubricant, for example, a count is kept. If the maximum is exceeded, the Autocycle Session will be shut down. This count is reset whenever a new Cycle Spec (default or transient) is applied or when a successful lubrication occurs. **Data Type:** Count
Cycles since activation	This is the number of cycles that have completed successfully since the Autocycle Session has been active, irrespective of what Cycle Spec has been in control and regardless of whether or not there have been any failed cycles. **Data Type:** Count
Lubricating	The Lubrication interval time remaining is meaningful only during this period (when this is set). **Data Type:** Boolean
Active	Whether or not the cycle is running. If not set, the Cycle time remaining value is meaningless. **Data Type:** Boolean
Deactivate	Whether or not the cycle is to be deactivated at the next available opportunity. **Data Type:** Boolean
Suspend requested	Whether or not the cycle has be requested to suspend. **Data Type:** Boolean
Wait time remaining	The amount of time to continue waiting in the case when lubrication was suspended. **Data Type:** Seconds

Autocycle Session Pycca attributes

```
<<Autocycle Session attributes>>=
attribute (Count Cycles_requested) default {0}
attribute (bool Continuous_operation) default {false}
attribute (Count Failed_cycles) default {0}
attribute (Count Cycles_since_activation) default {0}
attribute (bool Lubricating) default {false}
attribute (bool Active) default {false}
attribute (bool Deactivate) default {false}
attribute (bool Suspend_requested) default {false}
attribute (Seconds Wait_time_remaining) default {0}
```

R1 — Injector ⇒ Lubrication Schedule

- **Injector** is normally controlled by *exactly one* **Lubrication Schedule**
- **Lubrication Schedule** controls timing by default *zero or more* **Injector**

R1 Implementation

```
<<Injector references>>=
reference R1 -> Lubrication_Schedule
```

R2 — Lubrication Schedule ⇒ Injector

- **Autocycle Session** is an instance of **Lubrication Schedule** is timing control of *zero or more* **Injector**
- **Autocycle Session** is an instance of **Injector** is being controlled by *exactly one* **Lubrication Schedule**

Automatic lubrication is driven by a Lubrication Schedule which closely drives the timing of a single Injector. While multiple Injectors may be driven by the same Lubrication Specification, and hence the same relative time intervals, at a given moment each Injector will be in its own particular state and thus is driven by its own individual Autocycle Session.

All Injector control must, in fact, be cycle driven to ensure that the cycle is always synchronized correctly. To stop an Injector then, it is necessary to do it by deactivating its Autocycle Session.

R2 Implementation

```
<<Autocycle Session references>>=
reference R2_INJ -> Injector
reference R2_LBS -> Lubrication_Schedule

<<Injector references>>=
reference R2 -> Autocycle_Session
```

Machinery

The lubricant will be delivered to some type of equipment. This could be anything from a robot, to a jet engine to heavy lifting construction vehicle. From the perspective of the Lubrication domain, the actual function of the lubricated equipment is not relevant so it is represented as generic "machinery".

The particular values established for the Cycle and Injector Design attributes are determined, in part, by the specific needs and function of the lubricated Machinery.

ID {I}	This is an arbitrary value used for identification purposes only. **Data Type:** ID
Locked out	Sometimes the Machinery will be in an unsafe state or in a state where a technician is performing repairs on it and we want to ensure that no automated lubrication is active. This value is always checked to ensure that injection does not occur when the Machinery is isolated. **Data Type:** Boolean

Machinery Pycca attributes

```
<<Machinery attributes>>=
attribute (bool Locked_out) default {false}
```

Reservoir

Lubricant is stored in a small reservoir accessed by a single Injector. A sensor will report when it is full (normal) or low.

Injector {I, R3}	The lubricating Injector fueled by this Reservoir.

Level — The relative lubrication fluid level in the reservoir is tracked.
Data Type: Fluid State

Reservoir Pycca attributes

```
<<Reservoir attributes>>=
attribute (Fluid_State Level)
```

R3 — Injector ⇒ Reservoir

- **Injector** gets lubricant from *exactly one* **Reservoir**
- **Reservoir** supplies lubricant to *one or more* **Injector**

<explanation of how fluid is delivered and physical relationships . . . do some wiki research>

R3 Implementation

```
<<Reservoir references>>=
reference R3 ->>c Injector

<<Injector references>>=
reference R3 -> Reservoir
```

Domain Operations

Initialization

Because of the conditionality of R2 on the Lubrication Schedule side, it is necessary that any Injector instances defined in the initial instance population also have an instance of Autocycle Session. Examining the state model of Autocycle Session shows that the initial state is **Creating**. Normally, instances of Autocycle Session are created asynchronously and executing the action of the **Creating** causes them to transition to the **NOT_ACTIVE** state. However, initial instances are considered to be created synchronously and therefore the activity of the **Creating** state is not executed. At initialization time we must insure that the effect of the **Creating** state activity is executed. The definition of the initial instance insures that **Injector** and **Lubrication Schedule** instances are linked together. What we must do at initialization time is signal a **Created** event to each instance of **Autocycle Session** to drive them to the **NOT_ACTIVE** state.

```
<<domain operations>>=
domain operation
init()
{
    ClassRefVar(Autocycle_Session, acs) ;
    PYCCA_forAllInst(acs, Autocycle_Session) {
        if (IsInstInUse(acs)) {
            PYCCA_generate(Created, Autocycle_Session, acs, NULL) ;
        }
    }
}
```

Suspending an Autocycle Session

The Lube domain provides a function allowing a client to request the suspension of an **Autocycle Session**.

```
<<domain operations>>=
domain operation
Suspend_Autocycle_Session(
    InstId_t sessionId)
{
    PYCCA_checkId(Autocycle_Session, sessionId) ;
    ClassRefVar(Autocycle_Session, session) =
            PYCCA_refOfId(Autocycle_Session, sessionId) ;
    if (IsInstInUse(session)) {
        InstOp(Autocycle_Session, Suspend)(session) ;
    }
}
```

Injector Maximum Pressure

When the pressure on an **Injector** exceeds the maximum, then the Lube domain provides an interface to notify it of the condition.

```
<<domain operations>>=
domain operation
Injector_max_pressure(
    InstId_t injId)
{
    PYCCA_checkId(Injector, injId) ;
    ClassRefVar(Injector, inj) = PYCCA_refOfId(Injector, injId) ;
    if (IsInstInUse(inj)) {
        InstOp(Injector, Max_system_pressure)(inj) ;
    }
}
```

Machinery Locking

When the machinery is determined to have been locked out, this operation is invoked to inform the Lubrication domain of that fact.

```
<<domain operations>>=
domain operation
Lock_Machinery(
    InstId_t machineId)
{
    PYCCA_checkId(Machinery, machineId) ;
    ClassRefVar(Machinery, machine) = PYCCA_refOfId(Machinery, machineId) ;
    /*
     * Since the Machinery class has a static population, there is no
     * need to check if the instance is in use. All Machinery instances
     * defined in the initial instance population are always in use.
     */
    InstOp(Machinery, Lock)(machine) ;
}
```

Machinery Unlocking

When the machinery is determined to have been unlocked, this operation is invoked to inform the Lubrication domain of that fact.

```
<<domain operations>>=
domain operation
Unlock_Machinery(
    InstId_t machineId)
{
    PYCCA_checkId(Machinery, machineId) ;
    ClassRefVar(Machinery, machine) = PYCCA_refOfId(Machinery, machineId) ;
    InstOp(Machinery, Unlock)(machine) ;
}
```

External Operations

The Lubrication domain has a number of dependencies on entities outside of the domain. These dependencies are expressed as a set of external operations that request services. In `pycca` the code for the external operations is *not* included in the generated output. The code shown below is used by other tools that build a test harness for the domain.

User Interface Dependencies

```
<<external operations>>=
external operation
UI_Deactivated(
    InstId_t sessionId)
{
}
```

Application Error Log Dependencies

```
<<external operations>>=
external operation
App_Error(char const *Msg)
{
    fprintf(stderr, "%s\n", Msg) ;
}
```

Signal I/O Dependencies

```
<<external operations>>=
external operation
SIO_Inject(InstId_t injectorId)
{
}
```

```
<<external operations>>=
external operation
SIO_Stop_injecting(InstId_t injectorId)
{
}
```

```
<<external operations>>=
external operation
SIO_Start_monitoring(InstId_t injectorId)
{
}
```

```
<<external operations>>=
external operation
SIO_Stop_monitoring(InstId_t injectorId)
{
}
```

Alarm Dependencies

```
<<external operations>>=
external operation
ALARM_Set_pressure_error(InstId_t injectorId)
{
}
```

```
<<external operations>>=
external operation
ALARM_Set_dissipation_error(InstId_t injectorId)
{
}
```

```
<<external operations>>=
external operation
ALARM_Clear_dissipation_error(InstId_t injectorId)
{
}
```

```
<<external operations>>=
external operation
ALARM_Set_lube_level_very_low(InstId_t reservoirId)
{
}
```

```
<<external operations>>=
external operation
ALARM_Clear_lube_level_very_low(InstId_t reservoirId)
{
}
```

```
<<external operations>>=
external operation
ALARM_Set_lube_level_low(InstId_t reservoirId)
{
}
```

```
<<external operations>>=
external operation
ALARM_Clear_lube_level_low(InstId_t reservoirId)
{
}
```

```
<<external operations>>=
external operation
ALARM_Set_lube_level_empty(InstId_t reservoirId)
{
}
```

```
<<external operations>>=
external operation
ALARM_Clear_lube_level_empty(InstId_t reservoirId)
{
}
```

Active Classes

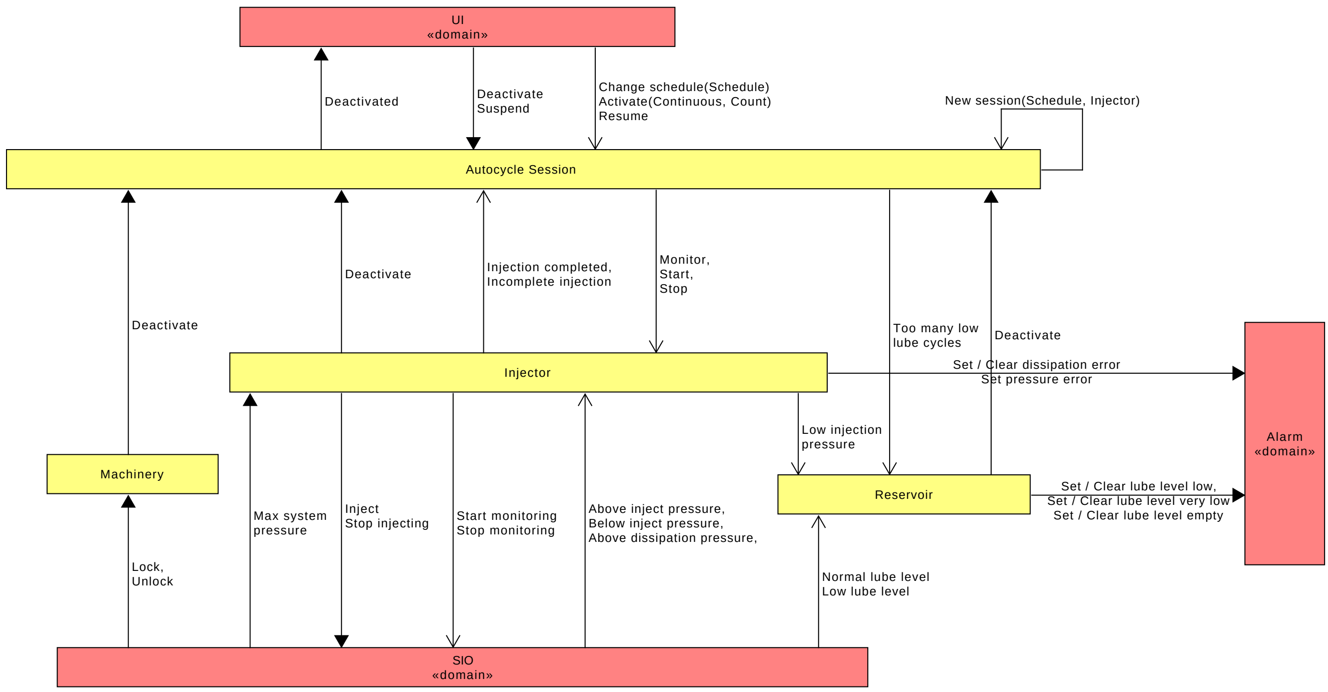

Figure 2: Collaboration Diagram

Autocycle Session State Model

Here is the Autocycle Session state diagram:

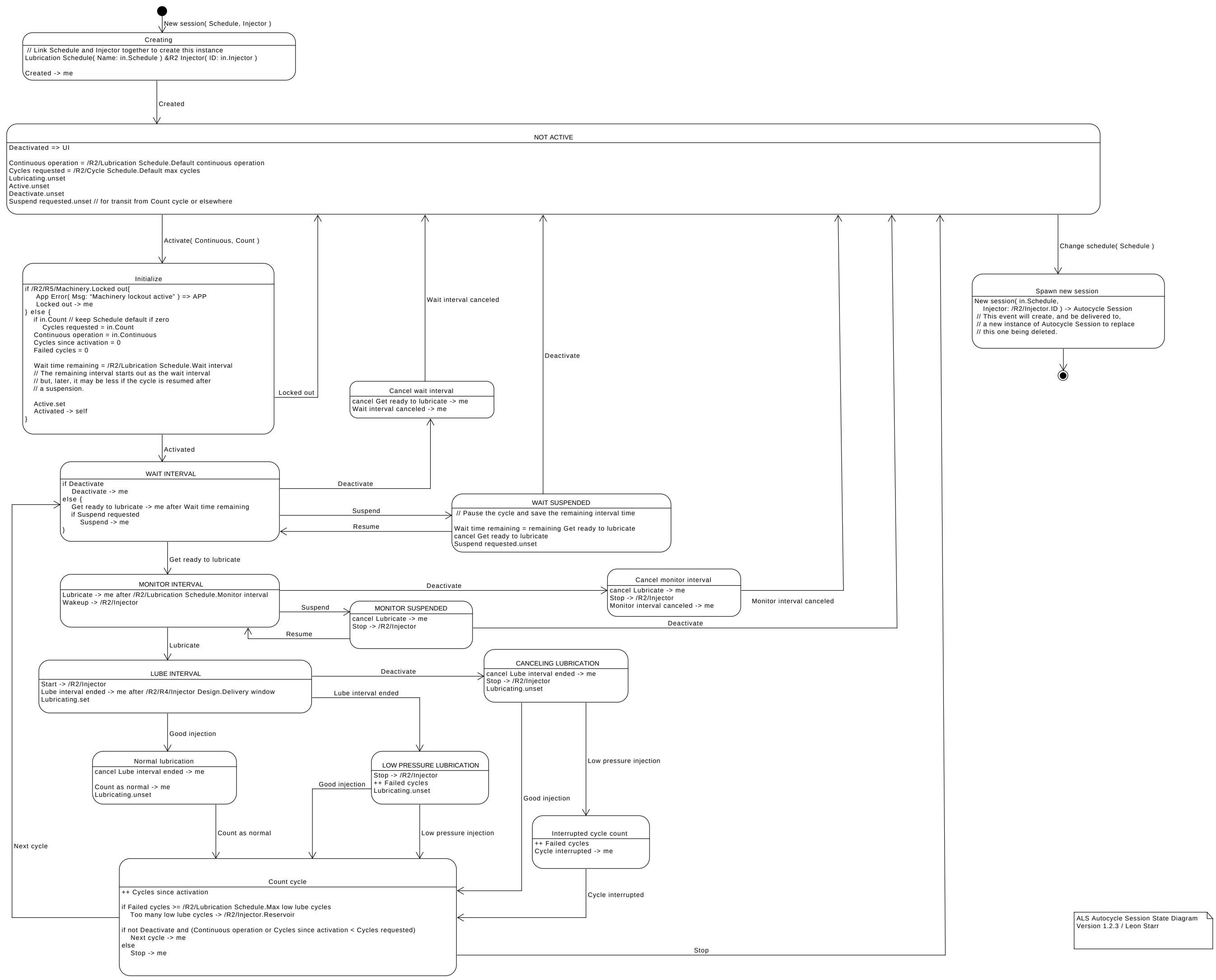

Figure 3: Autocycle Session State Diagram

```
<<Autocycle Session state model>>=
default transition CH
initial state Creating

transition . - New_session -> Creating

final state Spawn_new_session
```

Table 1: Autocycle Session wait states (non-local events)

	Change schedule	**Suspend**	**Resume**	**Activate**	**Deactivate**	**Low pressure injection**	**Good injection**
NOT ACTIVE	Spawning new session	IG	IG	Initialize	IG	CH	CH

Table 1: (continued)

	Change schedule	Suspend	Resume	Activate	Deactivate	Low pressure injection	Good injection
WAIT INTERVAL	IG	WAIT SUSPENDED	IG	IG	Cancel main interval	CH	CH
WAIT SUSPENDED	IG	IG	WAIT INTERVAL	IG	NOT ACTIVE	CH	CH
MONITOR INTERVAL	IG	MONITOR SUSPENDED	IG	IG	Cancel monitor interval	CH	CH
MONITOR SUSPENDED	IG	IG	MONITOR INTERVAL	IG	NOT ACTIVE	CH	CH
LUBE INTERVAL	IG	IG	IG	IG	CANCELING LUBRICATION	CH	Normal lubrication
LOW PRESSURE LUBRICATION	IG	IG	IG	IG	IG	Count cycle	Count cycle
CANCELING LUBRICATION	IG	IG	IG	IG	IG	Interrupted cycle count	Count cycle

```
<<Autocycle Session state model>>=
transition NOT_ACTIVE - Change_schedule -> Spawn_new_session
transition NOT_ACTIVE - Suspend -> IG
transition NOT_ACTIVE - Resume -> IG
transition NOT_ACTIVE - Activate -> Initialize
transition NOT_ACTIVE - Deactivate -> IG

transition WAIT_INTERVAL - Change_schedule -> IG
transition WAIT_INTERVAL - Suspend -> WAIT_SUSPENDED
transition WAIT_INTERVAL - Resume -> IG
transition WAIT_INTERVAL - Activate -> IG
transition WAIT_INTERVAL - Deactivate -> Cancel_wait_interval

transition WAIT_SUSPENDED - Change_schedule -> IG
transition WAIT_SUSPENDED - Suspend -> IG
transition WAIT_SUSPENDED - Resume -> WAIT_INTERVAL
transition WAIT_SUSPENDED - Activate -> IG
transition WAIT_SUSPENDED - Deactivate -> NOT_ACTIVE

transition MONITOR_INTERVAL - Change_schedule -> IG
transition MONITOR_INTERVAL - Suspend -> MONITOR_SUSPENDED
transition MONITOR_INTERVAL - Resume -> IG
transition MONITOR_INTERVAL - Activate -> IG
transition MONITOR_INTERVAL - Deactivate -> Cancel_monitor_interval

transition MONITOR_SUSPENDED - Change_schedule -> IG
transition MONITOR_SUSPENDED - Suspend -> IG
transition MONITOR_SUSPENDED - Resume -> MONITOR_INTERVAL
transition MONITOR_SUSPENDED - Activate -> IG
transition MONITOR_SUSPENDED - Deactivate -> NOT_ACTIVE
```

```
transition LUBE_INTERVAL - Change_schedule -> IG
transition LUBE_INTERVAL - Suspend -> IG
transition LUBE_INTERVAL - Resume -> IG
transition LUBE_INTERVAL - Activate -> IG
transition LUBE_INTERVAL - Deactivate -> CANCELING_LUBRICATION
transition LUBE_INTERVAL - Good_injection -> Normal_lubrication

transition LOW_PRESSURE_LUBRICATION - Change_schedule -> IG
transition LOW_PRESSURE_LUBRICATION - Suspend -> IG
transition LOW_PRESSURE_LUBRICATION - Resume -> IG
transition LOW_PRESSURE_LUBRICATION - Activate -> IG
transition LOW_PRESSURE_LUBRICATION - Deactivate -> IG
transition LOW_PRESSURE_LUBRICATION - Low_pressure_injection -> Count_cycle
transition LOW_PRESSURE_LUBRICATION - Good_injection -> Count_cycle

transition CANCELING_LUBRICATION - Change_schedule -> IG
transition CANCELING_LUBRICATION - Suspend -> IG
transition CANCELING_LUBRICATION - Resume -> IG
transition CANCELING_LUBRICATION - Activate -> IG
transition CANCELING_LUBRICATION - Deactivate -> IG
transition CANCELING_LUBRICATION - Low_pressure_injection -> Interrupted_cycle_count
transition CANCELING_LUBRICATION - Good_injection -> Count_cycle
```

Table 2: Autocycle Session wait states (delayed events)

	Lubricate	**Lube interval ended**	**Get ready to lubricate**
NOT ACTIVE	CH	CH	CH
WAIT INTERVAL	CH	CH	MONITOR INTERVAL
WAIT SUSPENDED	CH	CH	CH
MONITOR INTERVAL	LUBE INTERVAL	CH	CH
MONITOR SUSPENDED	CH	CH	CH
LUBE INTERVAL	CH	LOW PRESSURE LUBRICATION	CH
LOW PRESSURE LUBRICATION	CH	CH	CH
CANCELING LUBRICATION	CH	CH	CH

```
<<Autocycle Session state model>>=
transition WAIT_INTERVAL - Get_ready_to_lubricate -> MONITOR_INTERVAL

transition MONITOR_INTERVAL - Lubricate -> LUBE_INTERVAL

transition LUBE_INTERVAL - Lube_interval_ended -> LOW_PRESSURE_LUBRICATION
```

Table 3: Autocycle Session transitory states, part I

	Created	**Activated**	**Locked out**	**Next cycle**	**Stop**
Creating	NOT ACTIVE	CH	CH	CH	CH
Initialize	CH	WAIT INTERVAL	NOT ACTIVE	CH	CH
Normal lubrication	CH	CH	CH	CH	CH
Count cycle	CH	CH	CH	WAIT INTERVAL	NOT ACTIVE

Table 3: (continued)

	Created	Activated	Locked out	Next cycle	Stop
Cancel monitor interval	CH	CH	CH	CH	CH
Cancel wait interval	CH	CH	CH	CH	CH
Interrupted cycle count	CH	CH	CH	CH	CH

```
<<Autocycle Session state model>>=
transition Creating - Created -> NOT_ACTIVE

transition Initialize - Activated -> WAIT_INTERVAL
transition Initialize - Locked_out -> NOT_ACTIVE

transition Count_cycle - Next_cycle -> WAIT_INTERVAL
transition Count_cycle - Stop -> NOT_ACTIVE
```

Table 4: Autocycle Session transitory states, part II

	Count as normal	Monitor interval canceled	Wait interval canceled	Cycle interrupted
Creating	CH	CH	CH	CH
Initialize	CH	CH	CH	CH
Normal lubrication	Count Cycle	CH	CH	CH
Count cycle	CH	CH	CH	CH
Cancel monitor interval	CH	NOT ACTIVE	CH	CH
Cancel wait interval	CH	CH	NOT ACTIVE	CH
Interrupted cycle count	CH	CH	CH	Count cycle

```
<<Autocycle Session state model>>=
transition Normal_lubrication - Count_as_normal -> Count_cycle

transition Cancel_monitor_interval - Monitor_interval_canceled -> NOT_ACTIVE

transition Cancel_wait_interval - Wait_interval_canceled -> NOT_ACTIVE

transition Interrupted_cycle_count - Cycle_interrupted -> Count_cycle
```

Creating

Activity

```
 // Link Schedule and Injector together to create this instance
Lubrication Schedule( Name: in.Schedule ) &R2 Injector( ID: in.Injector )

Created -> me
```

Implementation

```
<<Autocycle Session state model>>=
state Creating(
```

```
    char const *schedule,
    unsigned injector)
{
    ClassRefVar(Lubrication_Schedule, ls) =
        ClassOp(Lubrication_Schedule, findByName)(rcvd_evt->schedule) ;
    assert(ls != NULL) ;
    ClassRefVar(Injector, inj) = PYCCA_refOfId(Injector, rcvd_evt->injector) ;
    self->R2_LBS = ls ;
    self->R2_INJ = inj ;
    inj->R2 = self ;

    PYCCA_generateToSelf(Created) ;
}
```

NOT ACTIVE

Activity

```
Deactivated => UI

Continuous operation = /R2/Lubrication Schedule.Default continuous operation
Cycles requested = /R2/Cycle Schedule.Default max cycles
Lubricating.unset
Active.unset
Deactivate.unset
Suspend requested.unset // for transit from Count cycle or elsewhere
```

Implementation

```
<<Autocycle Session state model>>=
state NOT_ACTIVE()
{
    ExternalOp(UI_Deactivated)(PYCCA_idOfSelf) ;

    ClassRefVar(Lubrication_Schedule, ls) = self->R2_LBS ;
    self->Continuous_operation = ls->Default_continuous_operation ;
    self->Cycles_requested = ls->Default_max_cycles ;
    self->Lubricating =
    self->Active =
    self->Deactivate =
    self->Suspend_requested = false ;
}
```

Initialize

Activity

```
if /R2/R5/Machinery.Locked out{
     App Error( Msg: "Machinery lockout active" ) => APP
     Locked out -> me
} else {
    if in.Count // keep Schedule default if zero
        Cycles requested = in.Count
    Continuous operation = in.Continuous
    Cycles since activation = 0
    Failed cycles = 0

    Wait time remaining = /R2/Lubrication Schedule.Wait interval
    // The remaining interval starts out as the wait interval
```

```
    // but, later, it may be less if the cycle is resumed after
    // a suspension.

    Active.set
    Activated -> self
}
```

Implementation

```
<<Autocycle Session state model>>=
state Initialize(
    bool continuous,
    Count count)
{
    if (self->R2_INJ->R5->Locked_out) {
        ExternalOp(App_Error)("Machinery lockout active") ;
        PYCCA_generateToSelf(Locked_out) ;
    } else {
        if (rcvd_evt->count != 0) {
            self->Cycles_requested = rcvd_evt->count ;
        }
        self->Continuous_operation = rcvd_evt->continuous ;
        self->Cycles_since_activation = 0 ;
        self->Failed_cycles = 0 ;

        self->Wait_time_remaining = self->R2_LBS->Wait_interval ;
        self->Active = true ;
        PYCCA_generateToSelf(Activated) ;
    }
}
```

WAIT INTERVAL

Activity

```
if Deactivate
    Deactivate -> me
else {
    Get ready to lubricate -> me after Wait time remaining
    if Suspend requested
        Suspend -> me
}
```

Implementation

```
<<Autocycle Session state model>>=
state WAIT_INTERVAL()
{
    if (self->Deactivate) {
        PYCCA_generateToSelf(Deactivate) ;
    } else {
        PYCCA_generateDelayedToSelf(Get_ready_to_lubricate,
            SecsToDelayTime(self->Wait_time_remaining)) ;
        if (self->Suspend_requested) {
            PYCCA_generateToSelf(Suspend) ;
        }
    }
}
```

Cancel wait interval

Activity

```
cancel Get ready to lubricate -> me
Wait interval canceled -> me
```

Implementation

```
<<Autocycle Session state model>>=
state Cancel_wait_interval()
{
    PYCCA_cancelDelayedToSelf(Get_ready_to_lubricate) ;
    PYCCA_generateToSelf(Wait_interval_canceled) ;
}
```

WAIT SUSPENDED

Activity

```
 // Pause the cycle and save the remaining interval time

Wait time remaining = remaining Get ready to lubricate
cancel Get ready to lubricate
Suspend requested.unset
```

Implementation

```
<<Autocycle Session state model>>=
state WAIT_SUSPENDED()
{
    self->Wait_time_remaining =
            DelayTimeToSecs(PYCCA_remainDelayedToSelf(Get_ready_to_lubricate)) ;
    PYCCA_cancelDelayedToSelf(Get_ready_to_lubricate) ;
    self->Suspend_requested = false ;
}
```

MONITOR INTERVAL

Activity

```
Lubricate -> me after /R2/Lubrication Schedule.Monitor interval
Wakeup -> /R2/Injector
```

Implementation

```
<<Autocycle Session state model>>=
state MONITOR_INTERVAL()
{
    PYCCA_generateDelayedToSelf(Lubricate,
        SecsToDelayTime(self->R2_LBS->Monitor_interval)) ;
    PYCCA_generate(Wakeup, Injector, self->R2_INJ, self) ;
}
```

Cancel monitor interval

Activity

```
cancel Lubricate -> me
Stop -> /R2/Injector
Monitor interval canceled -> me
```

Implementation

```
<<Autocycle Session state model>>=
state Cancel_monitor_interval()
{
    PYCCA_cancelDelayedToSelf(Lubricate) ;
    PYCCA_generate(Stop, Injector, self->R2_INJ, self) ;
    PYCCA_generateToSelf(Monitor_interval_canceled) ;
}
```

MONITOR SUSPENDED

Activity

```
cancel Lubricate -> me
Stop -> /R2/Injector
```

Implementation

```
<<Autocycle Session state model>>=
state MONITOR_SUSPENDED()
{
    PYCCA_cancelDelayedToSelf(Lubricate) ;
    PYCCA_generate(Stop, Injector, self->R2_INJ, self) ;
}
```

LUBE INTERVAL

Activity

```
Start -> /R2/Injector
Lube interval ended -> me after /R2/R4/Injector Design.Delivery window
Lubricating.set
```

Implementation

```
<<Autocycle Session state model>>=
state LUBE_INTERVAL()
{
    PYCCA_generate(Start, Injector, self->R2_INJ, self) ;
    PYCCA_generateDelayedToSelf(Lube_interval_ended,
            SecsToDelayTime(self->R2_INJ->R4->Delivery_window)) ;
    self->Lubricating = true ;
}
```

CANCELING LUBRICATION

Activity

```
cancel Lube interval ended -> me
Stop -> /R2/Injector
Lubricating.unset
```

Implementation

```
<<Autocycle Session state model>>=
state CANCELING_LUBRICATION()
{
    PYCCA_cancelDelayedToSelf(Lube_interval_ended) ;
    PYCCA_generate(Stop, Injector, self->R2_INJ, self) ;
    self->Lubricating = false ;
}
```

Normal lubrication

Activity

```
cancel Lube interval ended -> me

Count as normal -> me
Lubricating.unset
```

Implementation

```
<<Autocycle Session state model>>=
state Normal_lubrication()
{
    PYCCA_cancelDelayedToSelf(Lube_interval_ended) ;
    PYCCA_generateToSelf(Count_as_normal) ;
    self->Lubricating = false ;
}
```

LOW PRESSURE LUBRICATION

Activity

```
Stop -> /R2/Injector
++ Failed cycles
Lubricating.unset
```

Implementation

```
<<Autocycle Session state model>>=
state LOW_PRESSURE_LUBRICATION()
{
    PYCCA_generate(Stop, Injector, self->R2_INJ, self) ;
    self->Failed_cycles++ ;
    self->Lubricating = false ;
}
```

Count cycle

Activity

```
++ Cycles since activation

if Failed cycles >= /R2/Lubrication Schedule.Max low lube cycles
    Too many low lube cycles -> /R2/Injector.Reservoir

if not Deactivate and (Continuous operation or Cycles since activation < Cycles requested)
    Next cycle -> me
else
    Stop -> me
```

Implementation

```
<<Autocycle Session state model>>=
state Count_cycle()
{
    self->Cycles_since_activation++ ;
    if (self->Failed_cycles >= self->R2_LBS->Max_low_lube_cycles) {
        PYCCA_generate(Too_many_low_lube_cycles, Reservoir, self->R2_INJ->R3, self) ;
    }
    if (!self->Deactivate &&
            (self->Continuous_operation ||
            self->Cycles_since_activation < self->Cycles_requested)) {
        PYCCA_generateToSelf(Next_cycle) ;
    } else {
        PYCCA_generateToSelf(Stop) ;
    }
}
```

Interrupted cycle count

Activity

```
++ Failed cycles
Cycle interrupted -> me
```

Implementation

```
<<Autocycle Session state model>>=
state Interrupted_cycle_count()
{
    self->Failed_cycles++ ;
    PYCCA_generateToSelf(Cycle_interrupted) ;
}
```

Spawn new session

Activity

```
New session( in.Schedule,
    Injector: /R2/Injector.ID ) -> Autocycle Session
 // This event will create, and be delivered to,
 // a new instance of Autocycle Session to replace
 // this one being deleted.
```

Implementation

```
<<Autocycle Session state model>>=
state Spawn_new_session(
    char const *schedule)
{
    MechEcb new_session = PYCCA_newCreationEventForThisClass(New_session, self) ;
    PYCCA_eventParam(new_session, Autocycle_Session, New_session, schedule) =
            rcvd_evt->schedule ;
    PYCCA_eventParam(new_session, Autocycle_Session, New_session, injector) =
            PYCCA_idOfRef(Injector, self->R2_INJ) ;
    PYCCA_postEvent(new_session) ;

    self->R2_INJ->R2 = NULL ; // ❶
}
```

❶ NULL out the back link before we are deleted as a final state. Also, the count parameter doesn't seem to get used.

Autocycle Session Operations

```
<<Autocycle Session operations>>=
instance operation Deactivate() {
    self->Deactivate = true ;
    PYCCA_generateToSelf(Deactivate) ;
}
```

```
<<Autocycle Session operations>>=
instance operation Suspend() {
    self->Suspend_requested = true ;
    PYCCA_generateToSelf(Suspend) ;
}
```

Injector Class State Model

Here is the Injector state transition diagram:

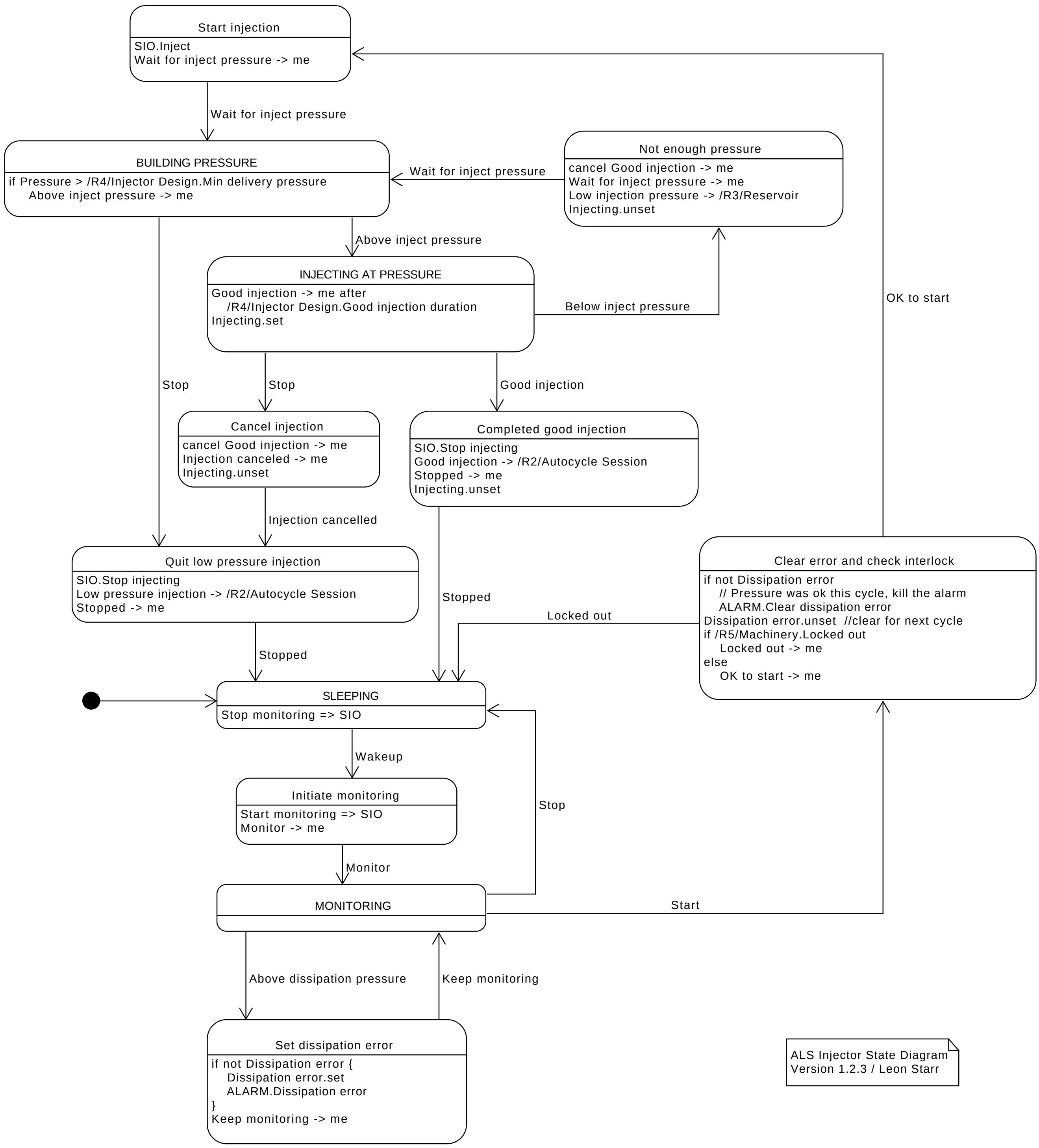

Figure 4: Injector State Diagram

```
<<Injector state model>>=
default transition CH
initial state SLEEPING
```

Table 5: Injector wait states

	Wakeup	Above inject pressure	Below inject pressure	Start	Stop	Above dissipation pressure	Good injection
BUILDING PRESSURE	CH	INJECTING AT PRESSURE	IG	CH	Quit low pressure injection	IG	CH
INJECTING AT PRESSURE	CH	IG	Not enough pressure	CH	Cancel injection	IG	Completed good injection
SLEEPING	Initiate monitoring	IG	IG	CH	IG	IG	CH
MONITORING	CH	IG	IG	Clear error and check interlock	SLEEPING	Set dissipation error	CH

```
<<Injector state model>>=
transition BUILDING_PRESSURE - Stop -> Quit_low_pressure_injection
transition BUILDING_PRESSURE - Above_inject_pressure -> INJECTING_AT_PRESSURE
transition BUILDING_PRESSURE - Below_inject_pressure -> IG
transition BUILDING_PRESSURE - Above_dissipation_pressure -> IG

transition INJECTING_AT_PRESSURE - Stop -> Cancel_injection
transition INJECTING_AT_PRESSURE - Good_injection -> Completed_good_injection
transition INJECTING_AT_PRESSURE - Below_inject_pressure -> Not_enough_pressure
transition INJECTING_AT_PRESSURE - Above_inject_pressure -> IG
transition INJECTING_AT_PRESSURE - Above_dissipation_pressure -> IG

transition SLEEPING - Wakeup -> Initiate_monitoring
transition SLEEPING - Above_inject_pressure -> IG
transition SLEEPING - Below_inject_pressure -> IG
transition SLEEPING - Stop -> IG
transition SLEEPING - Above_dissipation_pressure -> IG

transition MONITORING - Above_dissipation_pressure -> Set_dissipation_error
transition MONITORING - Start -> Clear_error_and_check_interlock
transition MONITORING - Stop -> SLEEPING
transition MONITORING - Above_inject_pressure -> IG
transition MONITORING - Below_inject_pressure -> IG
```

Table 6: Injector transitory states

	Wait for inject pressure	Monitor	Stopped	Locked out	Ok to start	Injection canceled	Keep monitoring
Start injection	BUILDING PRESSURE	CH	CH	CH	CH	CH	CH
Not enough pressure	BUILDING PRESSURE	CH	CH	CH	CH	CH	CH
Completed good injection	CH	CH	SLEEPING	CH	CH	CH	CH

Table 6: (continued)

	Wait for inject pressure	Monitor	Stopped	Locked out	Ok to start	Injection canceled	Keep monitor-ing
Quit low pressure injection	CH	CH	SLEEPING	CH	CH	CH	CH
Initiate monitoring	CH	MONITOR-ING	CH	CH	CH	CH	CH
Set dissipation error	CH	CH	CH	CH	CH	CH	MONITOR-ING
Clear error and check interlock	CH	CH	CH	SLEEPING	Start injection	CH	CH
Cancel injection	CH	CH	CH	CH	CH	Quit low pressure injection	CH

```
<<Injector state model>>=
transition Start_injection - Wait_for_inject_pressure -> BUILDING_PRESSURE

transition Not_enough_pressure - Wait_for_inject_pressure -> BUILDING_PRESSURE

transition Completed_good_injection - Stopped -> SLEEPING

transition Quit_low_pressure_injection - Stopped -> SLEEPING

transition Initiate_monitoring - Monitor -> MONITORING

transition Set_dissipation_error - Keep_monitoring -> MONITORING

transition Clear_error_and_check_interlock - Locked_out -> SLEEPING
transition Clear_error_and_check_interlock - OK_to_start -> Start_injection

transition Cancel_injection - Injection_canceled -> Quit_low_pressure_injection
```

Start injection

Activity

```
SIO.Inject
Wait for inject pressure -> me
```

Implementation

```
<<Injector state model>>=
state Start_injection()
{
    ExternalOp(SIO_Inject)(PYCCA_idOfSelf) ;
    PYCCA_generateToSelf(Wait_for_inject_pressure) ;
}
```

BUILDING PRESSURE

Activity

```
if Pressure > /R4/Injector Design.Min delivery pressure
     Above inject pressure -> me
```

Implementation

```
<<Injector state model>>=
state BUILDING_PRESSURE()
{
    if (self->Pressure > self->R4->Min_delivery_pressure) {
        PYCCA_generateToSelf(Above_inject_pressure) ;
    }
}
```

INJECTING AT PRESSURE

Activity

```
Good injection -> me after
    /R4/Injector Design.Good injection duration
Injecting.set
```

Implementation

```
<<Injector state model>>=
state INJECTING_AT_PRESSURE()
{
    PYCCA_generateDelayedToSelf(Good_injection,
            SecsToDelayTime(self->R4->Good_injection_duration)) ;
    self->Injecting = true ;
}
```

Not enough pressure

Activity

```
cancel Good injection -> me
Wait for inject pressure -> me
Low injection pressure -> /R3/Reservoir
Injecting.unset
```

Implementation

```
<<Injector state model>>=
state Not_enough_pressure()
{
    PYCCA_cancelDelayedToSelf(Good_injection) ;
    PYCCA_generateToSelf(Wait_for_inject_pressure) ;

    PYCCA_generate(Low_injection_pressure, Reservoir, self->R3, self) ;
    self->Injecting = false ;
}
```

Cancel injection

Activity

```
cancel Good injection -> me
Injection canceled -> me
Injecting.unset
```

Implementation

```
<<Injector state model>>=
state Cancel_injection()
{
    PYCCA_cancelDelayedToSelf(Good_injection) ;
    PYCCA_generateToSelf(Injection_canceled) ;
    self->Injecting = false ;
}
```

Completed good injection

Activity

```
SIO.Stop injecting
Good injection -> /R2/Autocycle Session
Stopped -> me
Injecting.unset
```

Implementation

```
<<Injector state model>>=
state Completed_good_injection()
{
    ExternalOp(SIO_Stop_injecting)(PYCCA_idOfSelf) ;
    PYCCA_generate(Good_injection, Autocycle_Session, self->R2, self) ;
    PYCCA_generateToSelf(Stopped) ;
    self->Injecting = false ;
}
```

Quit low pressure injection

Activity

```
SIO.Stop injecting
Low pressure injection -> /R2/Autocycle Session
Stopped -> me
```

Implementation

```
<<Injector state model>>=
state Quit_low_pressure_injection()
{
    ExternalOp(SIO_Stop_injecting)(PYCCA_idOfSelf) ;
    PYCCA_generate(Low_pressure_injection, Autocycle_Session, self->R2, self) ;
    PYCCA_generateToSelf(Stopped) ;
}
```

Clear error and check interlock

Activity

```
if not Dissipation error
    // Pressure was ok this cycle, kill the alarm
    ALARM.Clear dissipation error
Dissipation error.unset  //clear for next cycle
if /R5/Machinery.Locked out
    Locked out -> me
else
    OK to start -> me
```

Implementation

```
<<Injector state model>>=
state Clear_error_and_check_interlock()
{
    if (!self->Dissipation_error) {
        ExternalOp(ALARM_Clear_dissipation_error)(PYCCA_idOfSelf) ;
    }
    self->Dissipation_error = false ;

    if (self->R5->Locked_out) {
        PYCCA_generateToSelf(Locked_out) ;
    } else {
        PYCCA_generateToSelf(OK_to_start) ;
    }
}
```

SLEEPING

Activity

```
Stop monitoring => SIO
```

Implementation

```
<<Injector state model>>=
state SLEEPING()
{
    ExternalOp(SIO_Stop_monitoring)(PYCCA_idOfSelf) ;
}
```

Initiate monitoring

Activity

```
Start monitoring => SIO
Monitor -> me
```

Implementation

```
<<Injector state model>>=
state Initiate_monitoring()
{
    ExternalOp(SIO_Start_monitoring)(PYCCA_idOfSelf) ;
    PYCCA_generateToSelf(Monitor) ;
}
```

MONITORING

Activity

```
// empty activity
```

Implementation

```
<<Injector state model>>=
state MONITORING()
{
}
```

Set dissipation error

Activity

```
if not Dissipation error {
    Dissipation error.set
    ALARM.Dissipation error
}
Keep monitoring -> me
```

Implementation

```
<<Injector state model>>=
state Set_dissipation_error()
{
    if (!self->Dissipation_error) {
        self->Dissipation_error = true ;
        ExternalOp(ALARM_Set_dissipation_error)(PYCCA_idOfSelf) ;
    }
    PYCCA_generateToSelf(Keep_monitoring) ;
}
```

Injector Operations

```
<<Injector operations>>=
instance operation Max_system_pressure() {
    ExternalOp(ALARM_Set_pressure_error)(PYCCA_idOfSelf) ;
    ClassRefVar(Autocycle_Session, acs) = self->R2 ;
    InstOp(Autocycle_Session, Deactivate)(acs) ;
}
```

Reservoir State Model

Here is the Reservoir state diagram:

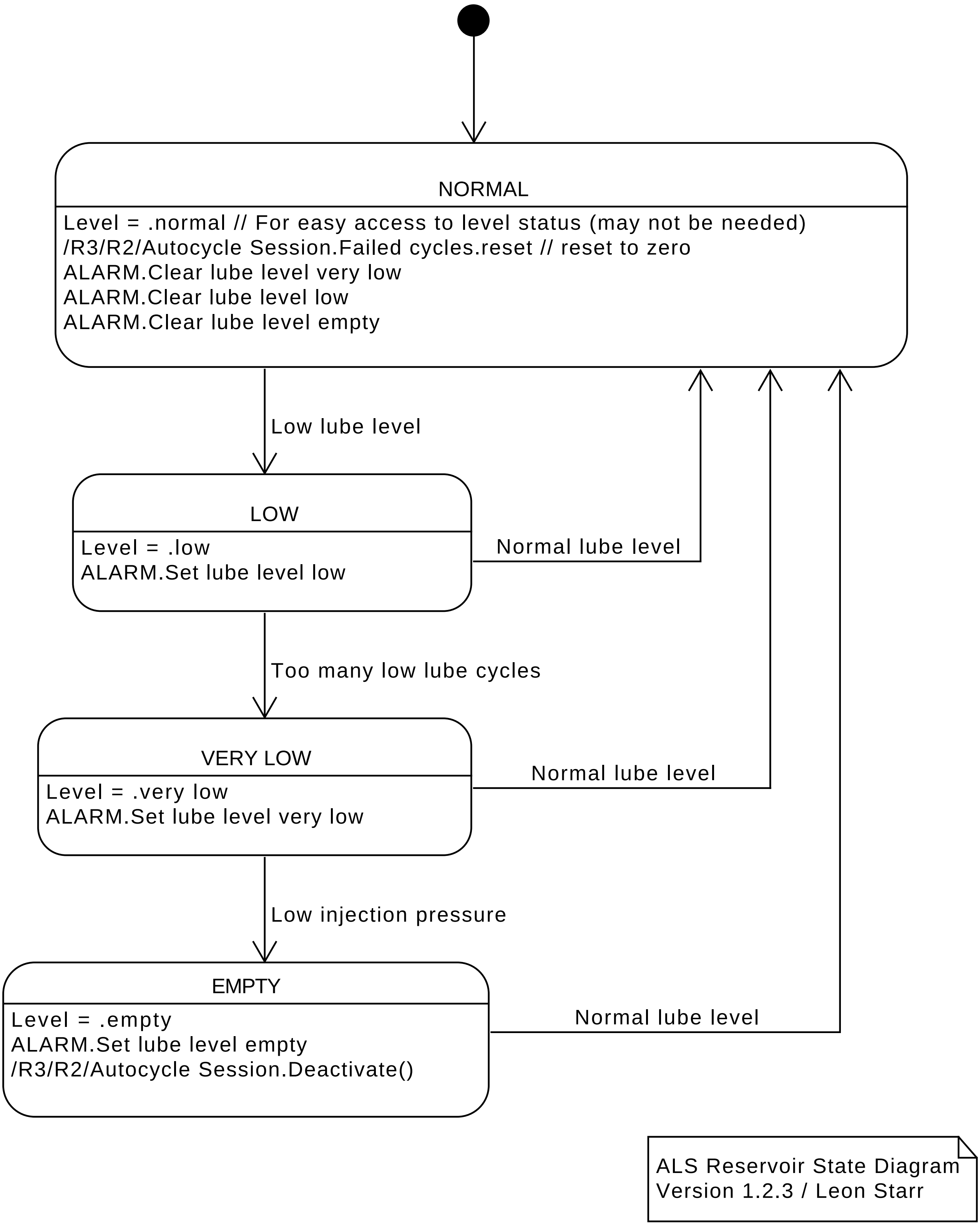

Figure 5: Reservoir State Diagram

```
<<Reservoir state model>>=
default transition CH
initial state NORMAL

transition NORMAL - Low_lube_level -> LOW
transition NORMAL - Normal_lube_level -> IG
transition NORMAL - Low_injection_pressure -> IG

transition LOW - Low_lube_level -> IG
transition LOW - Normal_lube_level -> NORMAL
transition LOW - Too_many_low_lube_cycles -> VERY_LOW
transition LOW - Low_injection_pressure -> IG

transition VERY_LOW - Low_lube_level -> IG
transition VERY_LOW - Normal_lube_level -> NORMAL
transition VERY_LOW - Low_injection_pressure -> EMPTY

transition EMPTY - Low_lube_level -> IG
transition EMPTY - Normal_lube_level -> NORMAL
transition EMPTY - Low_injection_pressure -> IG
```

NORMAL

Activity

```
Level = .normal // For easy access to level status (may not be needed)
/R3/R2/Autocycle Session.Failed cycles.reset // reset to zero
ALARM.Clear lube level very low
ALARM.Clear lube level low
ALARM.Clear lube level empty
```

Implementation

```
<<Reservoir state model>>=
state NORMAL()
{
    self->Level = FS_normal ;
    ClassRefConstSetVar(Injector, myinjs) ;
    PYCCA_forAllRelated(myinjs, self, R3) {
        ClassRefVar(Injector, inj) = *myinjs ;
        ClassRefVar(Autocycle_Session, acs) = inj->R2 ;
        acs->Failed_cycles = 0 ;
    }
    ExternalOp(ALARM_Clear_lube_level_very_low)(PYCCA_idOfSelf) ;
    ExternalOp(ALARM_Clear_lube_level_low)(PYCCA_idOfSelf) ;
    ExternalOp(ALARM_Clear_lube_level_empty)(PYCCA_idOfSelf) ;
}
```

LOW

Activity

```
Level = .low
ALARM.Set lube level low
```

Implementation

```
<<Reservoir state model>>=
state LOW()
{
    self->Level = FS_low ;
    ExternalOp(ALARM_Set_lube_level_low)(PYCCA_idOfSelf) ;
}
```

VERY LOW

Activity

```
Level = .very low
ALARM.Set lube level very low
```

Implementation

```
<<Reservoir state model>>=
state VERY_LOW()
{
    self->Level = FS_verylow ;
    ExternalOp(ALARM_Set_lube_level_very_low)(PYCCA_idOfSelf) ;
}
```

EMPTY

Activity

```
Level = .empty
ALARM.Set lube level empty
/R3/R2/Autocycle Session.Deactivate()
```

Implementation

```
<<Reservoir state model>>=
state EMPTY()
{
    self->Level = FS_empty ;
    ExternalOp(ALARM_Set_lube_level_empty)(PYCCA_idOfSelf) ;
    ClassRefConstSetVar(Injector, myinjs) ;
    PYCCA_forAllRelated(myinjs, self, R3) {
        ClassRefVar(Injector, inj) = *myinjs ;
        ClassRefVar(Autocycle_Session, acs) = inj->R2 ;
        InstOp(Autocycle_Session, Deactivate)(acs) ;
    }
}
```

Lubrication Schedule Operations

```
<<Lubrication Schedule operations>>=
class operation findByName(
    char const *name) : (struct Lubrication_Schedule *) {
    ThisClassRefVar(ls) ;
    PYCCA_selectOneStaticInstOfThisClassWhere(ls, strcmp(ls->Name, name) == 0)
    return ls == ThisClassEndStorage ? NULL : ls ;
}
```

Machinery Operations

```
<<Machinery operations>>=
instance operation Unlock() {
    self->Locked_out = false ;
}
```

```
<<Machinery operations>>=
instance operation Lock() {
    self->Locked_out = true ;
    ClassRefConstSetVar(Injector, myinjs) ;
    PYCCA_forAllRelated(myinjs, self, R5) {
        ClassRefVar(Injector, inj) = *myinjs ;
        ClassRefVar(Autocycle_Session, acs) = inj->R2 ;
        InstOp(Autocycle_Session, Deactivate)(acs) ;
    }
}
```

Initial Instance Population

Lubrication Schedule Population

Table 7: Lubrication Schedule Population

Name	Wait interval	Monitor interval	Max low lube cycles	Default continuous operation	Default max cycles
Shaft	90 s	30 s	10	true	10000
Gearbox	210 s	45 s	8	true	5000
Generator	120 s	25 s	10	true	10000
Test2	20 s	15 s	1	false	200

```
<<Lubrication Schedule population>>=
table
Lubrication_Schedule
    (Name_t Name)
    (Duration Wait_interval)
    (Duration Monitor_interval)
    (Count Max_low_lube_cycles)
    (bool Default_continuous_operation)
    (Count Default_max_cycles)

@gearbox    {"Gearbox"}     {210}   {45}    {8}     {true}  {5000}
@generator  {"Generator"}   {120}   {25}    {10}    {true}  {10000}
@shaft      {"Shaft"}       {90}    {30}    {10}    {true}  {10000}
@test2      {"Test2"}       {20}    {15}    {1}     {false} {200}
end
```

Autocycle Session Population

Table 8: Autocycle Session Population

Injector	Schedule	Cycles requested	Continuous operation	Failed cycles	Lubricating	Active	Deactivate	Wait time remaining
IN1	Gearbox	0	true	0	false	true	false	90 s
IN2	Shaft	0	true	1	true	true	true	0 s
IN3	Generator	0	true	0	false	true	true	0 s

```
<<Autocycle Session population>>=
table
Autocycle_Session
    R2_INJ
    R2_LBS
    (Count Cycles_requested)
    (bool Continuous_operation)
    (Count Failed_cycles)
    (bool Lubricating)
    (bool Active)
    (bool Deactivate)
    (Seconds Wait_time_remaining)

@acs1   -> in1 -> gearbox   {0} {true} {0} {false} {true} {false} {90}
@acs2   -> in2 -> shaft     {0} {true} {1} {true}  {true} {true}  {0}
@acs3   -> in3 -> generator {0} {true} {0} {false} {true} {true}  {0}
end
```

Injector Population

Table 9: Injector Population

ID	Pressure	Dissipation error	Injecting	Default schedule	Machinery	Reservoir	Model
IN1	20.3 MPa	false	false	Gearbox	M1	RES1	IX77B
IN2	0.0 MPa	false	false	Shaft	M2	RES2	IHN4
IN3	0.0 MPa	true	true	Gearbox	M3	RES1	IX77B

```
<<Injector population>>=
table
Injector
    (MPa Pressure)
    (bool Dissipation_error)
    (bool Injecting)
    R1
    R3
    R4
    R5
    R2

@in1    {20}    {false} {false} -> gearbox -> res1 -> ix77b -> m1 -> acs1
@in2    {0}     {false} {false} -> shaft -> res2 -> ihn4 -> m2 -> acs2
```

```
@in3    {0}     {true}  {true}  -> generator -> res1 -> ix77b -> m3 -> acs3
end
```

Injector Design Population

Table 10: Injector Design Population

Name	Min delivery pressure	Max system pressure	Max dissipation pressure	Delivery window	Good injection duration
IHN4	19 MPa	35 MPa	32 MPa	90 s	9 s
IX77B	15 MPa	26 MPa	26 MPa	120 s	11 s

```
<<Injector Design population>>=
table
Injector_Design
    (Model_Name Model)
    (MPa Min_delivery_pressure)
    (MPa Max_system_pressure)
    (MPa Max_dissipation_pressure)
    (Seconds Delivery_window)
    (Seconds Good_injection_duration)

@ihn4       {"IHN4"}    {19}    {35}    {32}    {90}    {9}
@ix77b      {"IX77B"}   {15}    {26}    {26}    {120}   {11}
end
```

Reservoir Population

Table 11: Reservoir Population

ID	Level
RES1	normal
RES2	low

```
<<Reservoir population>>=
table
Reservoir
    (Fluid_State Level)
    R3

@res1   {FS_normal} ->> in1 in3 end
@res2   {FS_low}    ->> in2 end
end
```

Machinery Population

Table 12: Machinery Population

ID	Locked out
M1	false
M2	false
M3	false

```
<<Machinery population>>=
table
Machinery
    (bool Locked_out)
    R5

@m1     {false}     ->> in1 end
@m2     {false}     ->> in2 end
@m3     {false}     ->> in3 end
end
```

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 usually the “C” file that is viewed in a debugger.

Root Chunk

```
<<lube.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:
```

```
#   Lubrication Domain Workbook
#
#*--

domain lube
    <<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>>=
<<Lubrication Schedule class>>
<<Injector Design class>>
<<Injector class>>
<<Autocycle Session class>>
<<Machinery class>>
<<Reservoir class>>
```

Lubrication Schedule Class

```
<<Lubrication Schedule class>>=
class Lubrication_Schedule
    <<Lubrication Schedule attributes>>
    <<Lubrication Schedule references>>
    <<Lubrication Schedule operations>>

    population static
end
```

Injector Design Class

```
<<Injector Design class>>=
class Injector_Design
    <<Injector Design attributes>>
end
```

Injector Class

```
<<Injector class>>=
class Injector
    <<Injector attributes>>
    <<Injector references>>
    machine
        <<Injector state model>>
    end
    <<Injector operations>>
end
```

Autocycle Session Class

```
<<Autocycle Session class>>=
class Autocycle_Session
    <<Autocycle Session attributes>>
    <<Autocycle Session references>>
    <<Autocycle Session operations>>
    machine
        <<Autocycle Session state model>>
    end
    population dynamic
end
```

Machinery Class

```
<<Machinery class>>=
class Machinery
    <<Machinery attributes>>
    <<Machinery references>>
    <<Machinery operations>>
end
```

Reservoir Class

```
<<Reservoir class>>=
class Reservoir
    <<Reservoir attributes>>
    <<Reservoir references>>
    machine
        <<Reservoir state model>>
    end
end
```

Population

```
<<population>>=
<<Lubrication Schedule population>>
<<Injector Design population>>
<<Injector population>>
<<Autocycle Session population>>
<<Machinery population>>
<<Reservoir population>>
```

Prologues

Implementation Prolog

```
<<implementation prolog>>=
implementation prolog {
    // Any additional implementation includes, etc.
    #include <assert.h>
    #include <time.h>
    #include <string.h>
    #include "lube.h"
    <<internal data types>>
    /*
     * To speed testing along, we will scale the time. When converting from
     * seconds to milliseconds in dealing with delayed events, we will us a
     * factor to allow us to scale real time. By default we will cause things
```

```
     * to run 4 times faster than real time.
     */
#ifdef INSTRUMENT
#   ifndef RUNFACTOR
#       define  RUNFACTOR 4UL
#   endif /* RUNFACTOR */
#   define SecsToDelayTime(s)   ((s) * (1000UL / RUNFACTOR))
#   define DelayTimeToSecs(d)   ((d) / (1000UL / RUNFACTOR))
#   else
#   define SecsToDelayTime(s)   ((s) * 1000UL)
#   define DelayTimeToSecs(d)   ((d) / 1000UL)
#endif /* INSTRUMENT */
}
```

Interface Prolog

```
<<interface prolog>>=
interface prolog {
    #include "pycca_portal.h"
    #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.