UML-B Exercise
Aircraft Engines

Exercise in UML-B State Machine Diagram modelling

Colin Snook

Exercise 3 - Aircraft Engines

An aircraft has one or more engines.

Each engine has 3 possible electrical power sources:

- An engine alternator which is only available when the engine is running,
- an airframe APU which must be used during starting,
- an airframe battery which is only sufficient to power electronic systems and to start the APU.

Engine start/run sequence:

- When the engine is not running it is either stood or being started.
- To start the engine, it is motored by an electrical starter motor until 'lightoff' is achieved.
- If lightoff is not achieved, the engine must be purged to remove unburnt fuel before it is returned to the stood state.
- If lightoff is achieved, the engine reaches ground idle and can be accelerated to flight idle.
- The engine must be returned to ground idle before it is shut off.
- To avoid damaging the alternator, power should be switched back to battery before shutting off the engine.

Exercise 3 - Aircraft Engines

Model this problem using UML-B state-machines attached to classes.

Build the model in stages using refinements and/or state-machine nesting as appropriate.

Check the model using the state-machine animator and the Pro-B model checker.

Context Diagram

Firstly, we model the relationship between Aircraft and Engines.
An Engine belongs to exactly one Aircraft (also known as Airframe, hence Af).

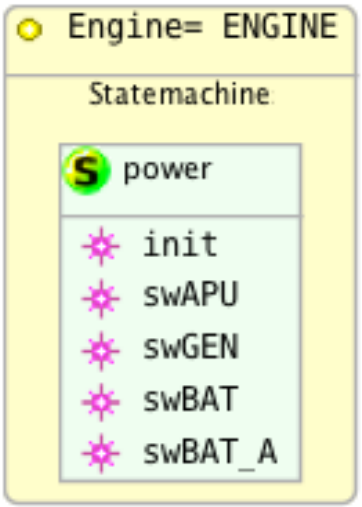
(Most of the behaviour concerns the Engines, therefore we choose the association direction so that it becomes an attribute of Engines since this is where we are most likely to refer to this relationship).

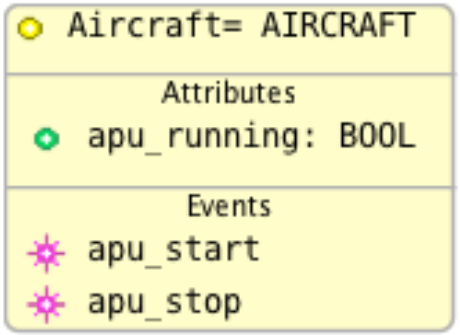
Class Diagram

We introduce classes and link their (fixed*) instances to the corresponding Class Types. (*We do not need to model creation of Aircraft or Engines).

The APU is a feaure of the airframe so we need to model it in the Aircraft class. We are only concerned whether it is running or not, so a boolean attribute and events to start and stop it are sufficient.

We also model the Engines power switching at this level (in a statemachine). We will leave the engine starting etc. for a later refinement.



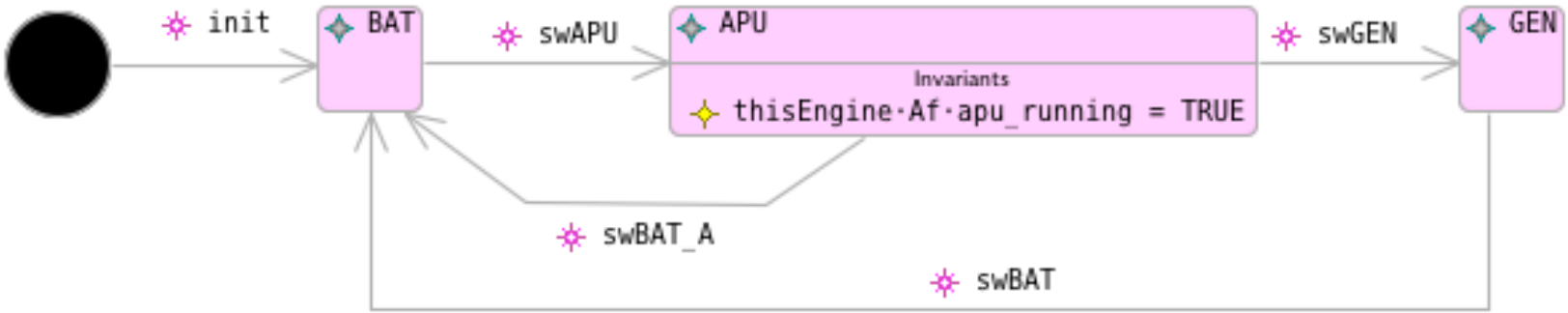


State-machine for power selection

This is the statemachine modelling the power source selector for an Engine. We initialise to BAT (battery) mode. From there we can switch to APU and then to GEN (generator) or back to BAT. From GEN we can switch back to BAT.

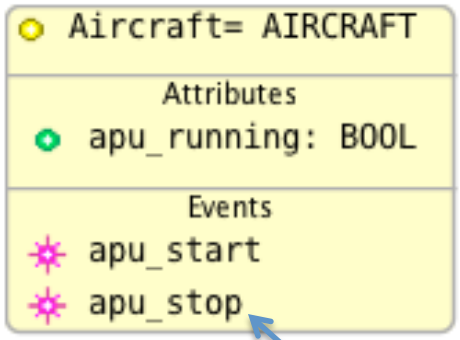
We should not use APU power unless the APU is running so we add an invariant to express this in the APU state. To make sure the invariant isn't violated, we also add a guard to prevent switching to this state unless the APU is running.

Of course, this is not sufficient as shown on the next page.



Guard for stopping an APU

We also need to ensure that the APU is not stopped while it is being used by an engine. (Note that this guard will need to be implemented as some form of interlock in the real system).



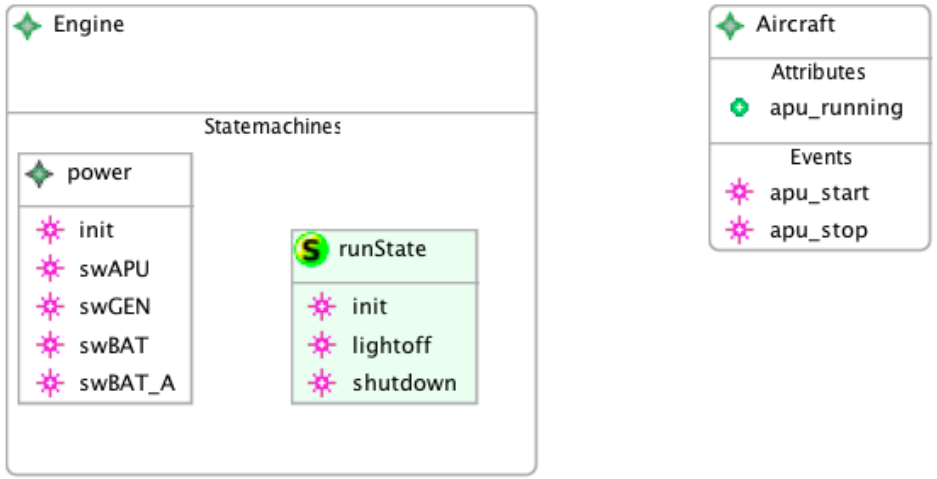
The guard refers to the statemachine (power) in the Engine class. (Note that the guard assumes the state-function translation is used). In the guard, the function is domain restricted to the Engines that are linked to this Aircraft and range restricted to those that are using APU power. If this leaves an empty set then there are no Engines for this Aircraft that are using APU power.

guard: $Af\sim[\{thisAircraft\}] \triangleleft power \triangleright \{APU\} = \varnothing$

First Refinement – Introducing Engine State

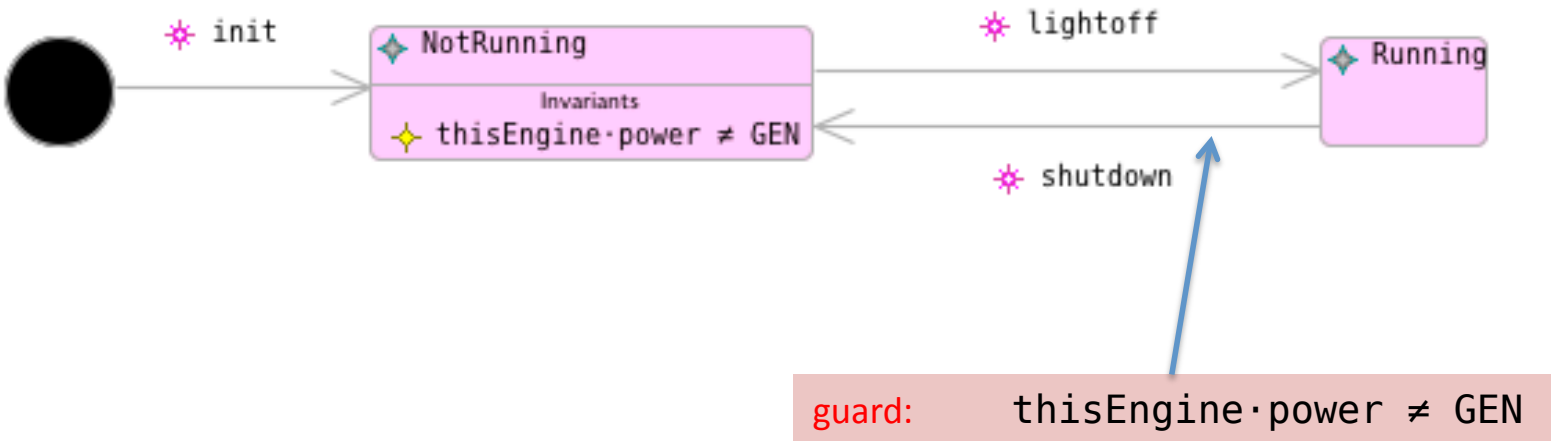
In this refinement, we start to model the run state of an engine.

We created a starting point for the refinement using the ‘make refinement’ button in the package diagram. This gives us refined Classes corresponding to those in our previous machine. We can then add a new statemachine to the Engine class to model its ‘runState’.



Statemachine for runState

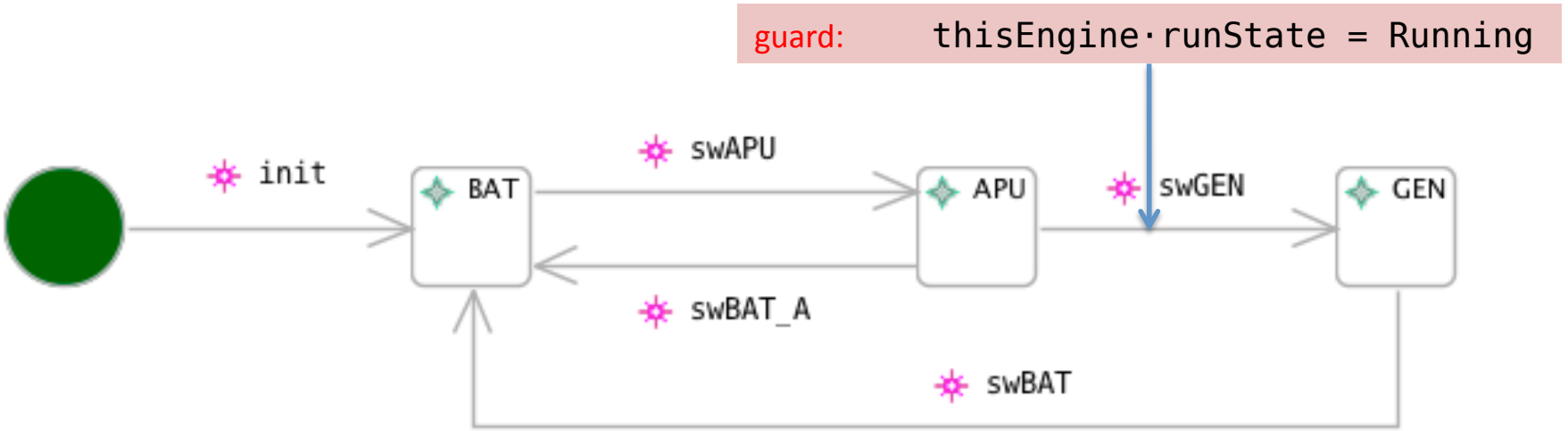
For this stage we will only introduce enough detail to handle issues about using the generator. All we need for this is to know whether the engine is running or not. If the engine is not running, we should not be using the generator so we add an invariant to the NotRunning state to express this, and a guard to ensure we do not shutdown while the generator is still selected.



Guard for switching to generator power

Having added details that reveal when we can and can't use the generator we now need to restrict when we are allowed to select it. To do this we add a guard to swGEN.

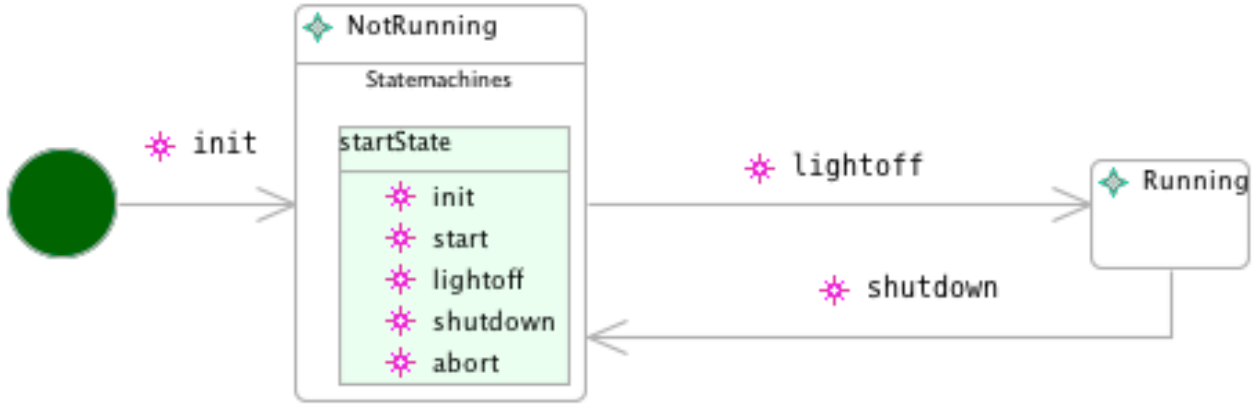
(Note that we are allowed to strengthen the guards of existing events in refinements. By doing so, we refine the behaviour to be closer to what we want in the final system).



Second Refinement

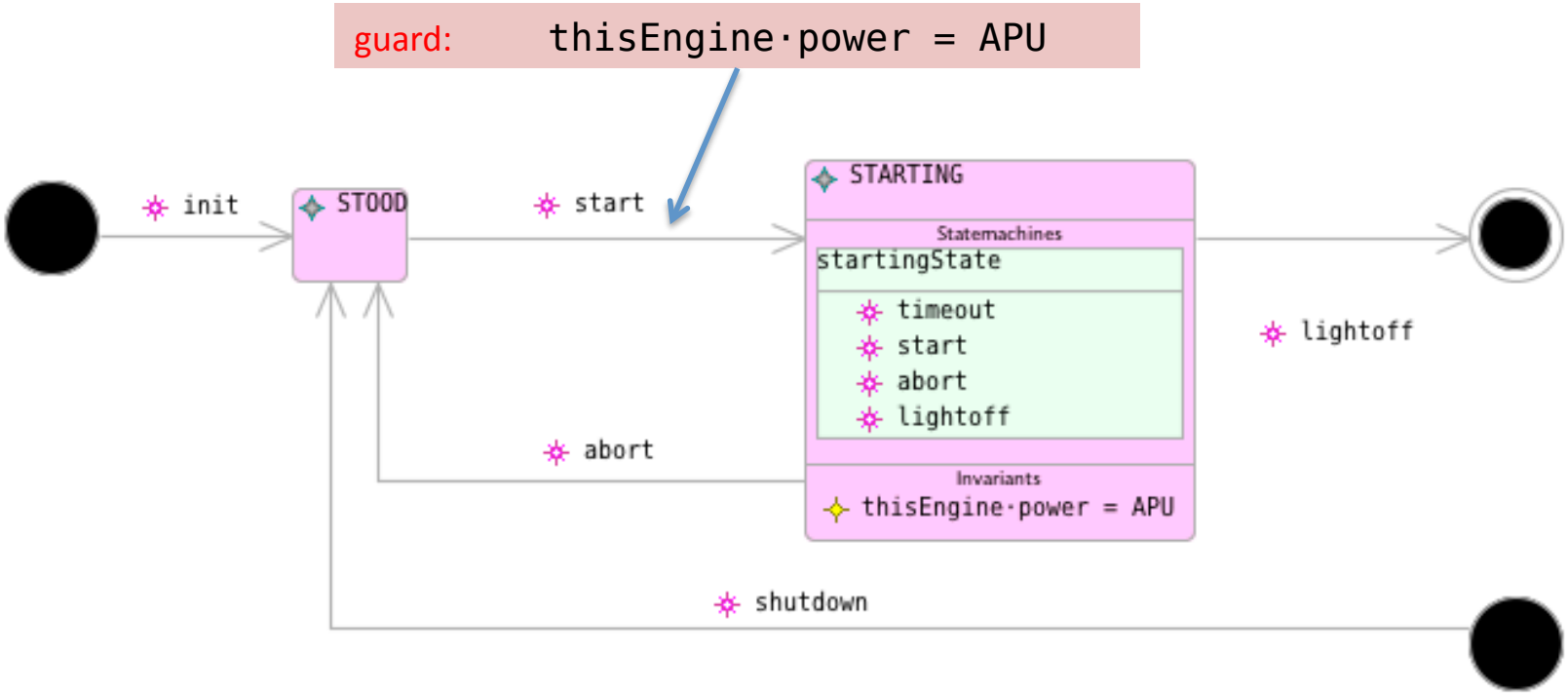
For this stage we will add details about attempting to start an engine. We can then deal with the constraints on using the APU as a power source.

We add a nested statemachine to the NotRunning State to model the startState.



Statemachine for starting

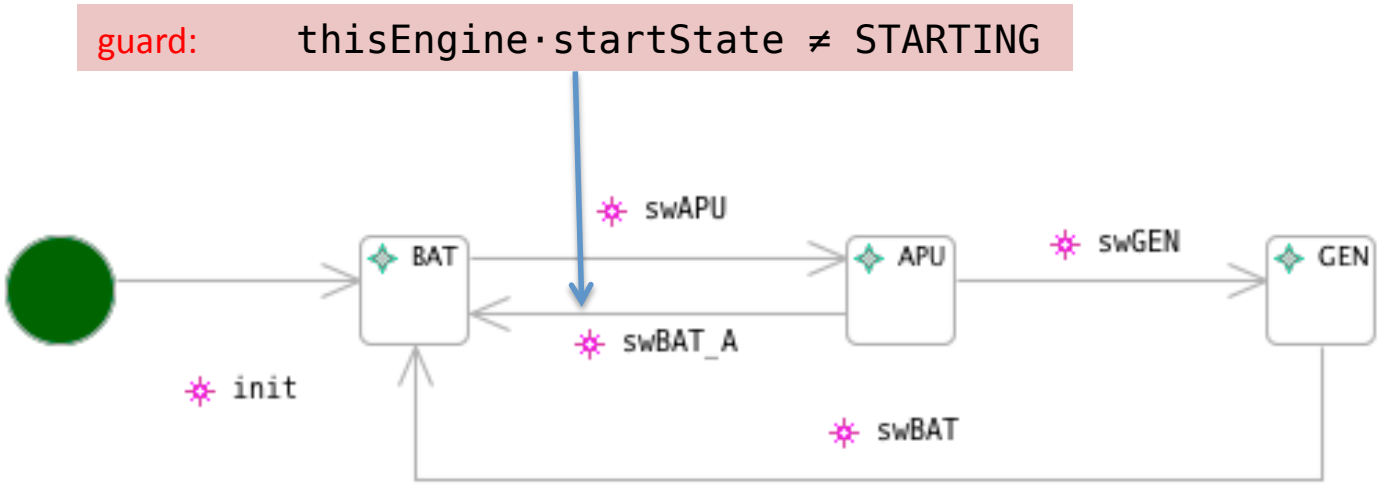
The Engine is initially STOOD and can then be started. Once it is STARTING the start may either achieve a sucessful lightoff or may be aborted. We add the details about aborting a start in a nested statemachine. This enables us to add the invariant that the APU must be selected while attempting to start. Note that incoming and outgoing transitions elaborate those of the parent state



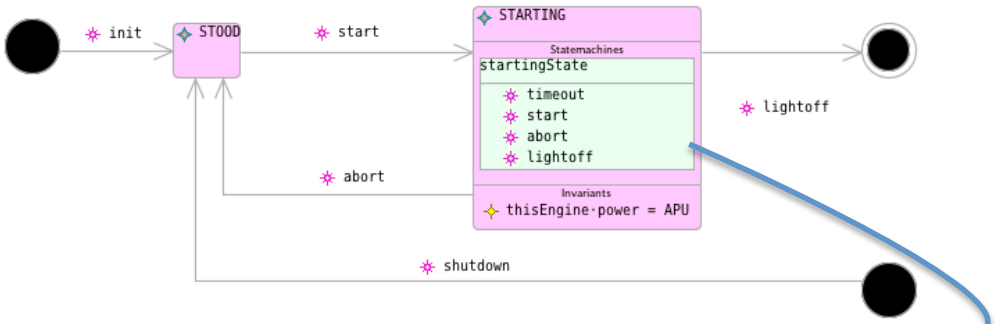
Guard for switching back to battery power

To preserve the invariant we also need to add a guard to prevent switching back to battery power while the engine is starting.

Note that we do not need to add a guard to swGEN because the one we added before, to ensure the engine is running implies the engine is no longer starting.

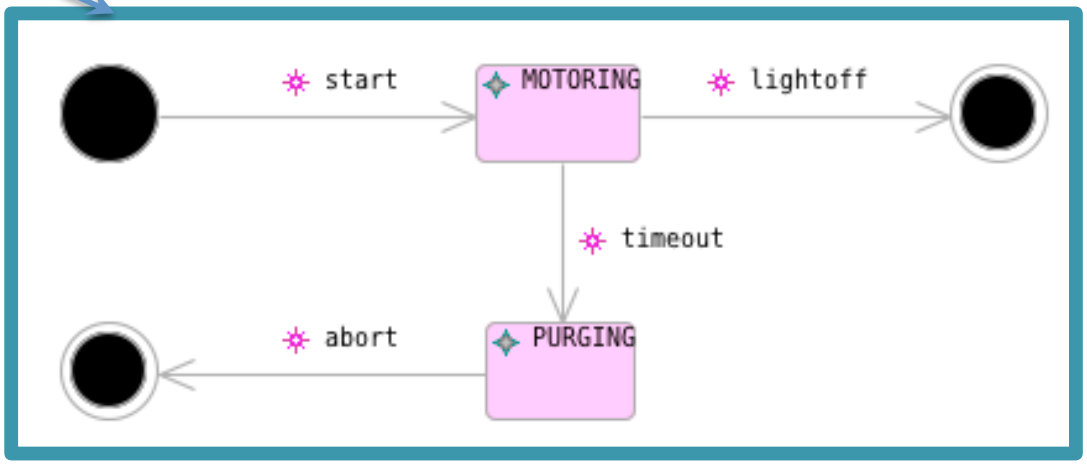


Elaborating the Starting process



The starting process is elaborated by adding a statemachine startingState inside STARTING. This addition is quite simple because it doesn't depend on any other feature such as the power source so we decided that a separate refinement wasn't necessary.

Another option would have been to put these details in the parent statemachine (i.e. without nesting). However, this would have made it more difficult to express the invariant for the STARTING state



Third refinement – elaborating the run state

The third refinement is also quite simple. We just need to add a nested statemachine to the run state in order to model the engine being accelerated to flight idle and back. There is no further dependency on the power source.