## Third refinement

Define the scope of this **model and the concerned requirements.**

The final layer of refinement takes into account the notion of clearances, authorizations and the scenarios of runways being closed off due to construction work, vehicle occupancy etc.

4 additional events are defined-

1. Accept\_clearance: parametric event enables to associate a plane with a runway (as one action); the other action adds this runway to rwy\_occ. This event basically gives the go ahead to any plane wanting to enter a particular runway and books the runway.
2. Add\_rwy\_nok: parametric event enables to add a not cleared runway to rwy\_nok. In the event of a runway being closed off, this event prevents any plane from getting authorization for that runway and consequently, the clearance.
3. Free\_rwy\_occ: parametric event enables to remove a pair of (plane, runway) from the clearance list (as one action); the other action removes this runway from rwy\_occ. This event is triggered when an aircraft leaves the runway. It no longer has clearance for that runway and the runway is open to other aircraft.
4. Free\_rwy\_nok: parametric event enables to remove a runway from rwy\_nok. This event is triggered when a closed off runway is ready to resume operation.

Describe the Context and the **Machine by giving pictures of those components.**

**CONTEXT:**

A new set RUNWAYS is introduced which holds all the runways of the airport.

**MACHINE:**

In the final refinement, every movement to and from the runway is dictated by the clearance which is a variable set that associates one plane to one runway at a time. The variable *aut* relates the set PLANES to the set RUNWAYS. In other words, it maintains all possible clearances that any plane can have.

The clearance is checked for the events where the plane is going on the runway while it is removed when the plane leaves the runway.

**Enter\_rwy & Land** : Clearance is checked

**Leav\_rwy & Takeoff** : Clearance is removed

Everytime a plane leaves a runway, it has to ask for a new clearance the next time it needs to access a runway. Clearance is provided through the Accept\_clearance event:

**Accept\_clearance**: This event relates a particular plane to a particular runway while making sure no other plane has clearance for the same runway. It also books the runway for the plane by adding it to the rwy\_occ set.

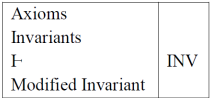
**Add\_rwy\_nok**: In the event a runway is not suitable for operation, this event moves it to the rwy\_nok set. It removes a clearance for the runway (if any exist) and blocks the runway.

**Free\_rwy\_occ**: Everytime a plane leaves a runway, it has to be made available for other possible planes. This event removes the plane from the rwy\_occ set.

**Free\_rwy\_nok**: This event removes a runway form the rwy\_nok set after it has been cleared for operation.

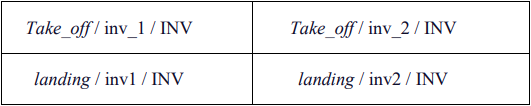
***Remark***: If Accept\_clearance event is triggered, the planes in the taxiway (if any) are given priority for clearance over planes in the air wanting to land.

Discuss the validation of the **PO invariant rules and the DLF rule (without the variant topic).**

A proof obligation is something that has to be proven to show the consistency of the machine, the correctness of theorems, etc. A proof obligation consists of a label, a number of hypothesis that can be used in the proof and a goal – a predicate that must be proven.

**Sequent:**

A sequent is a formal statement describing something we want to prove. Sequents are of the following form where H is the set of hypotheses (predicates) and G is the goal that can be proved from the predicates. The statement can be read as follows: Under the hypotheses H, prove the goal G.

The abstract model has 2 invariants and 2 events. Therefore, a total of 4 Proof Obligation rules need to be applied.

**Sequence – 1**

**Take\_off / inv1/ INV:**

|  |
| --- |
| nb\_max ϵ ℕ  nb\_max = 20  nb\_ground ϵ ℕ  nb\_ground ≤ nb\_max  Ⱶ  nb\_ground – 1 ϵ ℕ |

Axiom **axm\_1**

Axiom **axm\_2**

Invariant **inv\_1**

Invariant **inv\_2**

Ⱶ

Modified Invariant **inv\_1**

After applying the proof –

|  |
| --- |
| nb\_max ϵ ℕ  nb\_max = 20  nb\_ground ϵ ℕ  nb\_ground ≤ nb\_max  Ⱶ  nb\_ground – 1 ϵ ℕ |

|  |
| --- |
| nb\_ground ≤ nb\_max  Ⱶ  nb\_ground – 1 ϵ ℕ |

MON

MON stands for monotonicity of hypotheses. Through the MON hypothesis rule, the unwanted hypothesis will be taken out, leaving only the necessary hypothesis.

**Sequence - 2**

**Take\_off / inv2 / INV:**

|  |
| --- |
| nb\_max ϵ ℕ  nb\_max = 20  nb\_ground ϵ ℕ  nb\_ground ≤ nb\_max  Ⱶ  nb\_ground – 1 ≤ nb\_max |

Axiom **axm\_1**

Axiom **axm\_2**

Invariant **inv\_1**

Invariant **inv\_2**

Ⱶ

Modified Invariant **inv\_1**

After applying the proof –

|  |
| --- |
| nb\_max ϵ ℕ  nb\_max = 20  nb\_ground ϵ ℕ  nb\_ground ≤ nb\_max  Ⱶ  nb\_ground – 1 ≤ nb\_max |

|  |
| --- |
| nb\_ground ≤ nb\_max  Ⱶ  nb\_ground-1 ≤ nb\_max |

MON

**Sequence - 3**

**Landing / inv1 / INV:**

|  |
| --- |
| nb\_max ϵ ℕ  nb\_max = 20  nb\_ground ϵ ℕ  nb\_ground ≤ nb\_max  Ⱶ  nb\_ground + 1 ϵ ℕ |

Axiom **axm\_1**

Axiom **axm\_2**

Invariant **inv\_1**

Invariant **inv\_2**

Ⱶ

Modified Invariant **inv\_1**

After applying the proof –

|  |
| --- |
| nb\_max ϵ ℕ  nb\_max = 20  nb\_ground ϵ ℕ  nb\_ground ≤ nb\_max  Ⱶ  nb\_ground + 1 ϵ ℕ |

|  |
| --- |
| nb\_ground ϵ ℕ  Ⱶ  nb\_ground + 1 ϵ ℕ |

MON P2

**Sequence - 4**

**Landing / inv2 / INV:**

|  |
| --- |
| nb\_max ϵ ℕ  nb\_max = 20  nb\_ground ϵ ℕ  nb\_ground ≤ nb\_max  Ⱶ  nb\_ground + 1 ≤ nb\_max |

Axiom **axm\_1**

Axiom **axm\_2**

Invariant **inv\_1**

Invariant **inv\_2**

Ⱶ

Modified Invariant **inv\_1**

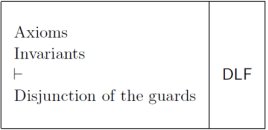
After applying the proof –

|  |
| --- |
| nb\_max ϵ ℕ  nb\_max = 20  nb\_ground ϵ ℕ  nb\_ground ≤ nb\_max  Ⱶ  nb\_ground + 1 ≤ nb\_max |

|  |
| --- |
| nb\_ground ≤ nb\_max  Ⱶ  nb\_ground + 1 ≤ nb\_max |

MON

**Deadlock Freedom:**

The DLF rule exists to prevent the possibility of a scenario occurring when no events can be triggered, i.e there is no state transition that can occur. The rule ensures that atleast one set of guard conditions of the events is always fulfilled.

|  |
| --- |
| nb\_max ϵ ℕ  nb\_max = 20  nb\_ground ϵ ℕ  nb\_ground ≤ nb\_max  Ⱶ  nb\_ground > 0 V nb\_ground < nb\_max |

Axiom **axm\_1**

Axiom **axm\_2**

Invariant **inv\_1**

Invariant **inv\_2**

Ⱶ

Disjunction of the guards **grd1 V grd2**

Through this rule, one of the guard conditions will always remain true.

# Conclusion

*Give a synthesis of this formal specification by using Event-B, and personal feedback.*

*Propose possible future works.*

The Event-B is a notation and method developed from the B-Method and is intended to be used with an incremental style of modelling. Event-B method relates to the specification, design, and coding of the associated software, and implements a proof framework. Thus, it makes it possible to carry out a development of correct systems by construction. To master the complexity, the specifications are represented using abstract models based on set theory and the logic of first-order predicates. Successive refinements make it possible to detail these models, which results in more concrete modelling. In the automated flight controller, we had the restriction to not allow take-off and landing of the aircraft at the same time but addition of manifold requirements and refinements can allow us to pursue the adverse thus making more complex in reality.

**References**

1. Rodin User’s Handbook v.2.8, Michael Jastram.

2. Chaudemar. J.C. « Basic Introduction to Systems Engineering ». 2018