Validation of the ABZ Landing Gear System using ProB

Dominik Hansen, Lukas Ladenberger, Harald Wiegard, Jens Bendisposto, Michael Leuschel

Universität Düsseldorf[**]
Institut für Informatik,
Universitätsstr. 1, D-40225 Düsseldorf
{hansen, ladenberger, wiegard, bendisposto, leuschel}@cs.uni-duesseldorf.de

Abstract. In this paper we present our formalisation of the ABZ landing gear case study in Event-B. The development was carried out using the Rodin platform and mainly used superposition refinement to structure the specification. To validate the model we complemented proof with animation and model checking. For the latter, we used the ProB animator and model checker. Graphical representation of the model turned out to be crucial in the development and validation of the model; this was achieved using a new version of BMotion Studio integrated into ProB 2.0.

1 Introduction

The "classical" B-method [1] and its successor the Event-B method [2] are refinement based formal methods. While the B-method is geared towards software development, the Event-B method is more tailored towards systems modelling. Refinement can be used to structure the development and proofs, and allows introducing complexity gradually. In Event-B the concept of refinement has been considerably extended: events can added, extended, split up or merged, parameters can be refined, removed or added, witnesses can be provided for automatic refinement proofs, event termination (convergence) can be proven or delayed to other refinement layers, etc. Model structuring, on the other hand, is much simpler in Event-B than in classical B: at one particular refinement level an Event-B model consists of a main machine which contains variables and events and a series of contexts which contain constants and sets. Composition and decomposition notions have been developed [9] for Event-B, but are not part of the core Event-B method and we have not used them in our case study.

In this paper, we present our results and experiences in formalising and validating the ABZ case study in Event-B. To carry out the study, we have chosen

[**] The work in this paper is partly funded by ADVANCE, an European Commission Information and Communication Technologies FP7 project.

the Rodin [3], ProB [8] and BMotion Studio [7] tools. The tools are tightly integrated. Rodin enables the use of the Event-B method and supports rigorous reasoning by formal proving. ProB can animate and modelcheck Event-B models, as well as provides a lot of features, for instance the inspection of the desired behaviour of a model. BMotion Studio is a framework for creating visualizations for formal models.

Structure of the Paper. Section 2 describes the Event-B model of the landing gear system and its refinement hierarchy. In Section 3 we demonstrate different approaches to validate our model. Section 4 describes the graphical representation of the model and outlines its use and benefits. Finally, Section 5 contains the conclusion and discusses future work.

Additional Material

For more information and resources, we refer the reader to our website:
`http://stups.hhu.de/ProB/index.php5/ABZ14`.
The website contains the model, visualization, and video material.

2 The Event-B Model and its Refinement Hierarchy

In this section, we describe our Event-B model of the landing gear system. This section may give the impression that the development was a linear process; in reality we started several times from scratch and adapted prior refinements. Initially, we also experimented with a classical B model, where structuring is easier but the stricter refinement concept makes environment modelling more difficult. We specify both the digital part and the environment in the same Event-B model. Hence, we obtain a integrated view of the system where we try to make a clear separation between both parts. Each event is associated either with the digital part or with the environment. The same applies to the variables, except for some shared variables (sensors and digital orders) which are used for the interaction of both parts. Figure 1 shows the interaction of the digital part and its environment. The figure may look like a clone of a picture from the original case-study specification, but it is actually part of the interactive visualization of our model.

2.1 Door and Landing Gear

We start by specifying the mechanical part of the landing gear system. The top-level abstraction of our development only model one door and one landing gear. The door is represented as a variable with the following possible states:

- closed,
- door_moving, and
- open.

The states of the landing gear variable are:

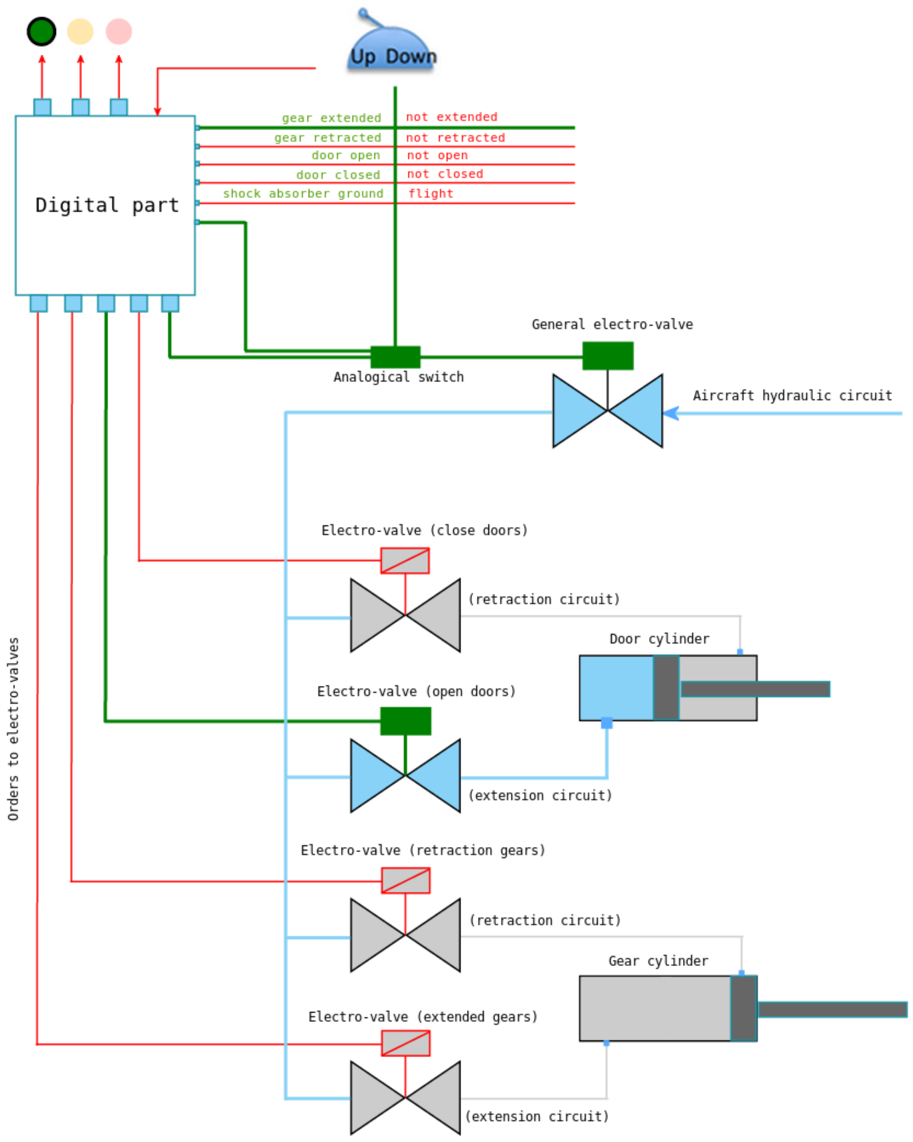


Fig. 1. Interaction of the digital part and the environment

– retracted,
– gear_moving, and
– extended.

In addition, we define events to change the states of these variables as illustrated in figure 2. Thus, at this point we only forbid direct jumps from closed (or retracted) to open (or extended) or back. We do not define a fixed sequence of states, and the behaviours of the gear and the door are fully independent of each other.

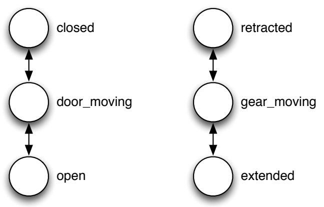


Fig. 2. Possible Door and Landing Gear States and Transitions

2.2 Electro-valves

In the first refinement step we add hydraulic elements to our specification. For each valve except for the general valve[1] we add a variable with the possible states *valve_open* and *valve_closed*. We define events to change the state of each valve. Moreover, we connect the behaviour of the door and the gear to the states of the corresponding valves. This is done by adding additional guards to the events that change state of the door and the gear. For example, the event that moves the door from the closed position to the moving positing can only be executed if the variable *open_door_valve* has the state *valve_open*. No movement of the door/gear is possible if two contrary valves (e.g. open_door_valve and close_door_valve) are simultaneously open. In this case the door/gear remains in its current position.

2.3 Outputs of the digital part

Next, we connect the electric orders of the digital part to the electro-valves of the hydraulic part. The electric orders are modelled as boolean variables. In this refinement we only regard the electric orders stimulating the the electro-valves introduced in the last refinement step. We add events to change the states of electric orders. Now, we only allow a behaviour of the electric-valves if the corresponding orders come from the digital part. Moreover we forbid stimulating two

[1] The general valve is introduced in a later refinement step.

contrary orders (open door/close door, extend/retract gear) simultaneously by adding additional guards to the events. We specify the requirement R_4 explicitly as these two invariants:

```
invariants
   R41:  ¬(open_EV = TRUE  ∧ close_EV = TRUE)
   R42:  ¬(retract_EV = TRUE  ∧ extend_EV = TRUE)
```

Note that the electric orders only enable events that change the states of the valves and do not execute them. This is important in order to later introduce failing electro-valves.

2.4 Controller Sensors

In this refinement step we introduce the input sensors of the digital part. We abstract the sensors by only considering the common value of the different channels. We add the following sensors:

- door_closed
- door_open
- gear_extended
- gear_retracted

The sensors are set by the events which change the state of the door and the gear. They directly reflect the state of the mechanical part but it would also be possible to introduce new events to update the sensor states according to their mechanical counterpart. After this refinement the controller only allows moving orders to the gear when the door is open, i.e. the *door_open* sensor is TRUE (R_{31}). Analogously, the controller only allows moving orders for the door when the gear is extended or retracted (R_{32}). These requirements are modelled as guards.

2.5 Controller Behaviour

In this refinement we define the sequence of output orders produced by the digital part. We abstract the digital part to consist of one controller producing the synthesized outputs of both computing modules. The sequence executed by the controller depends on the handle that can be moved by the pilot. We add a variable representing the handle and an event to change the state of the handle from *up* to *down* and vice versa. The handle event can be executed at any time. Moreover, the shock absorber sensor affects the behaviour of the controller. There must not be an order to retract the gear if the shock absorber is not relaxed, i.e., the aircraft is landed.

It is not difficult to define the uninterrupted outgoing and retraction sequence. However, allowing an interruption of the sequences by a counter order of the handle at any time makes this refinement step complex and tricky. We managed this by using some additional internal variables and adding guards to the events which change the states of the output orders. For example, the electric order towards the retraction valve can not occur if the handle is in the down position (R_{21}) and the order towards the extension valve can not occur if the handle is in the up position (R_{22}).

2.6 Analogical switch and general electro-valve

The analogical switch is intended to prevent the hydraulic part against abnormal behaviour. We add a variable for analogical switch and two events to change its state from *open* to *closed* and back. Moreover, we need an internal controller variable to record a handle movement and add this boolean variable as guard to the event which closes the switch. The general electro-valve is needed to supply the other electro-valves with hydraulic power from the aircraft hydraulic circuit. We add a variable for the electrical order (*general_EV*) coming from the digital part and events to change its state. The order for the general electro-valve must occur before the controller can produce an order to the other electro-valves. Hence, we add guards to all events which stimulate the other electro-valves. The following invariants ensure this behaviour:

invariants

$R511$: $open\_EV = TRUE \implies general\_EV = TRUE$
$R512$: $close\_EV = TRUE \implies general\_EV = TRUE$
$R513$: $extend\_EV = TRUE \implies general\_EV = TRUE$
$R514$: $retract\_EV = TRUE \implies general\_EV = TRUE$

Furthermore we add a variable for the general electro-valve and events to change it state. A behaviour of the gear/door can only occur if the general electro-valve and the corresponding maneuvering valve is open.

2.7 Cockpit lights

Besides the orders to the hydraulic part, the controller produces three further output signals to the cockpit:
- gears_locked_down
- gears_maneuvering
- anomaly

We add three boolean variables to represent these output signals and events to change it state. The two signals representing the state of gear can be easily computed by regarding the input sensors from the mechanical part. The *gears_locked_down* output directly correspond to the *gear_extended* input sensor. The *gears_maneuvering* output equals TRUE if the *gear_extended* and *gear_retracted* sensors are both FALSE.

In this refinement we abstract all possible inconsistent behaviours by one *anomaly* variable. We introduce a single event to set the variable to TRUE (without any guards). This event represents that the controller has detected an inconsistent behaviour.

In addition to the signals produced by the controller, we add three variables to represent the signal lights in the cockpit. The events to switch the lights on/off are connected to the output signals of the controller.

2.8 Further refinement steps

Several further refinement steps are needed to cover the complete specification of the landing gear system. For example, a further refinement step should introduce time and timing constraints. We experimented by introducing discrete time and an environment event (tick) incrementing the time. This approach works well from a theoretical point of view. Whenever an event with a time-based requirement is executed, the current time is saved in a designated variable. When the handle is pushed up, the current time will be saved to the variable $timerHandleUp$. This variable will be set to -1 if the handle is moved down. This allows us to formulate liveness conditions such as the requirement R_1 (stronger version) as an invariant:

```
invariants
  R11s:  anomaly = FALSE ∧ timerHandleUp>−1∧
         time ≥ timerHandleUp+150  ⟹
         gear_retracted = TRUE ∧ door_closed = TRUE
```

However, in practice, it is very complex to introduce timer for each timing constraint of the specification. Hence, we did not finished this refinement step due to the lack of time. Another refinement step should break up the abstractions we did so far (e.g. triplicating the sensors). We stopped at this point by getting a sufficient model to control the graphical visualization we made. Moreover, our model allows us to validate some of the “normal mode” requirements of the specification.

3 Validating the Model

This section describes validations carried out using ProB. In that setting the graphical visualization of the landing gear system was important. The latter is described separately in Section 4.

3.1 Invariants

As already mentioned, the requirements R_4 and R_5 are specified as invariants on different refinements levels. Our approach to validate an invariant is as follows: Before proving the invariants, we always run the model checker ProB. Sometimes the model checker provides us with a counter-example violating an invariant. In such cases we revisited and fixed our model by adding or modifying some guards. Moreover, we used another feature of ProB which is called constrained based model checking. In this mode of operation, ProB does not explore all reachable states starting from the initial state(s), but checks whether applying a single operation can result in a invariant independently of the particular initialization of the Event-B machine. If the constraint based checker finds a counter-example, this indicates that the model may contain a problem. The sequence of operations discovered by the constraint based checker leads from a valid state (satisfying the invariant) to a invariant violation, meaning that the B machine may have to be corrected. The valid state is not necessarily reachable from a valid initial state. However, this situation indicates that it will be impossible to prove the machine using the Event-B proof rules.

If ProB does not provide a counter-example we start proving the invariants. For the invariants specified in our final model the Rodin's provers are able to automatically discharge all generated proof obligations.

3.2 Animation and Model Testing

The animation feature of ProB had a major impact on our modelling process. Each time we added some new events to our Event-B model we ran the animator to check the new behaviour. To validate the complex behaviour of the controller (Sec. 2.5) we automated this approach. We used the animator to create valid traces (sequence of executed events) of the controller interacting with the environment. For example, we animated the complete outgoing and retraction sequence by letting the environment react in regular way. Additionally, we create traces by interrupting both sequences at each position by a counter order of the handle. All traces were saved and used as regression tests to validate further modifications of the model.

3.3 Temporal formulas

The requirements R_{11} and R_{12} (weaker version) describe a temporal behaviour of the system. The desired goal is to show that if the handle is pushed up/down the end of the retraction/extension sequence will be reached. Normally we would write such a liveness condition using the following simple LTL pattern:

$$\Box(t \Rightarrow \Diamond g)$$

where t is a trigger (handle movement) and g is the goal state which should be finally reached (gear are retracted/extened and the door is closed). However on the path from the handle movement to the end of the corresponding sequence several conditions must to be ensured. For example the handle must stay in its position and no anomaly should occur. To ensure these conditions we use a more complex LTL pattern:

$$\Box(t \Rightarrow ((g \; R \, s) \Rightarrow \Diamond g))$$

In this pattern, we only regard the paths (after the handle movement) that satisfy the condition s. The condition s must be satisfied only until the goal state is reached. Therefore the goal predicate releases (R) the condition s. Note that the release operator does not require that the goal state is finally reached. For example, the whole LTL formula for R_{12} is as follows:

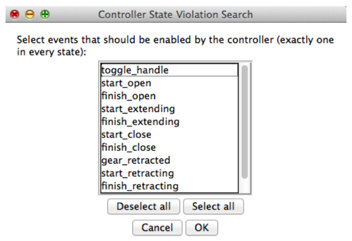
$$\begin{array}{l}
\Box(handle = up \Rightarrow \\
\qquad (((gear\_retracted = TRUE \wedge door\_closed = TRUE) \\
\qquad\quad R \;\; (handle = up \wedge gear\_shock\_absorber = flight \wedge anomaly = false)) \\
\qquad\qquad \Rightarrow \Diamond \; gear\_retracted = TRUE \wedge door\_closed = TRUE))
\end{array}$$

In contrast to invariants, LTL formulas are not automatically satisfied by further refinement steps and we have to re-check them at each level. Sometimes this requires some additional conditions such as fairness for certain events. We use the LTL model checker of ProB to validate the LTL formulas for the requirements R_{11} and R_{12}.

3.4 Relative Deadlock Freedom and Determinism Checking

The classical deadlock notion is not very useful for our model as the environment contains events that are always enabled. Instead we developed and used a new feature of ProB to check if a controller event is always possible. The feature is called relative deadlock checking and is able to only regard a certain selection of events. We checked the refinement described in Sect. 2.3 for relative deadlock freedom. In this refinement step, the controller does not have to wait for an environment behaviour, hence a controller event should always be possible.

Another important point is that the controller should behave in a deterministic way. In our model the controller behaviour is divided into several events. Therefore we have to ensure that for the same inputs the controller always produces the same outputs. To verify this, we developed and used another new feature of ProB checking that only one controller event is enabled at the same time (see Figure 3). More formally, the user selects a set of events $e_1, \ldots, e_k$ and the ProB model checker verifies that for every reachable state exactly one event e_i is enabled. We believe that this feature will be of interest for other Event-B system developments. In particular, it would have been handy for the case study reported in [6].


Controller State Violation Search

Select events that should be enabled by the controller (exactly one in every state):

toggle_handle
start_open
finish_open
start_extending
finish_extending
start_close
finish_close
gear_retracted
start_retracting
finish_retracting

Fig. 3. Controller Violation Search Dialog

4 Graphical Visualization

To visualize our model we used the new version of BMotion studio in ProB 2.0. We have not yet released ProB 2.0 officially, but the source code is available from [4]. Nightly builds are available from a Rodin updatesite [5]. One of the main differences between the current ProB Plug-in for Rodin and ProB 2.0 is that the latter is no longer based on Eclipse but rather uses standard Browser technology as its GUI. This allows to integrate ProB into a wide range of tools. Rodin is one of them but it is also possible to integrate ProB into a regular website or a presentation tool.

Another very important difference between ProB 2.0 and its predecessor is the tight integration with the Groovy scripting language. ProB 2.0 is implemented as if it was a library for Groovy. Basically everything from the constraint solver to the user interface is exposed to the scripting language. This makes it very easy to programatically control ProB 2.0.

The graphical interface consists of HTML, CSS and optionally JavaScript. This makes the user interface very flexible and compositional. We can design and implement each bit of the application separately and compose them in ways that are almost arbitrary. For the Rodin integration we bundle information into components, that are then displayed inside an Eclipse view. For instance, one component consists of the the list of events that can be executed in a given state and some control buttons to execute random animation steps and to go back and forth in time. This component is shown in the Events view within Rodin.

The new version of BMotion Studio [7] is just a view like every other view. It translates a state into a graphical representation. Once an animation is started, BMotion Studio is notified about every state change and then updates the graphical representation according to the state of the animation.

Originally BMotion Studio [7] was developed as a separate plug-in for Rodin. It used the Eclipse Graphical Editing Framework (GEF) to provide an editor to create visualizations of a model. While this was a very convenient approach to create simple visualizations, the visual editor makes it hard to create complex visualizations. For instance, creating a large table or a railroad track layout is very cumbersome.

BMotion Studio for ProB 2.0 follows the same principles as ProB 2.0. Most parts of BMotion Studio are accessible via the Groovy scripting language. Additionally the user interface can make use of JavaScript. This makes ist much easier to create complex or dynamic visualizations. A track layout, for example, can be created by any graphic program that is able to produce a SVG vector graphic. One advantage of the SVG format is that it is very easy to manipulate graphical components.

4.1 Visualization of the Landing Gear System

Our landing gear visualization consists of two parts, a graphical part, and an observer part. Simple SVG widgets, like shapes represent the different aspects of the architecture of the hydraulic part of the landing gear system as shown in Figure 1. The observer part acts as the link between the formal model and the graphical part. It defines expressions and predicates written in B that are evaluated by ProB in the current state of the simulation. The results of the expressions and predicates are used by the observers to update the visualization. For instance, the colour of the lines that represent the electric orders to the elector-valves is switched from red to green and from green to red whenever the corresponding variable is set to true or false respectively. This is shown in Figure 1, where the electric order to the open door electro valve is coloured in green whereas the other electric orders are coloured in red. The blue coloured lines represent the current circulation of pressure.

Another example is the position of the door cylinder: It is shifted in respect of the state of the door (*closed*, *door_moving* and *open*). In Figure 1 the door cylinder illustrates the *door_moving* state. We also created a separate view of the physical environment as shown in Figure 4 that represents beside the state of the cylinders, the current state of the physical door and gear.

The strict separation into a graphical part and an observer part makes the visualization reusable for other Event-B or even Classical B models of the landing gear system. Indeed, to adapt the visualizations for a different formal model one simply has to change the the observer part of the visualization, but not the graphical part.

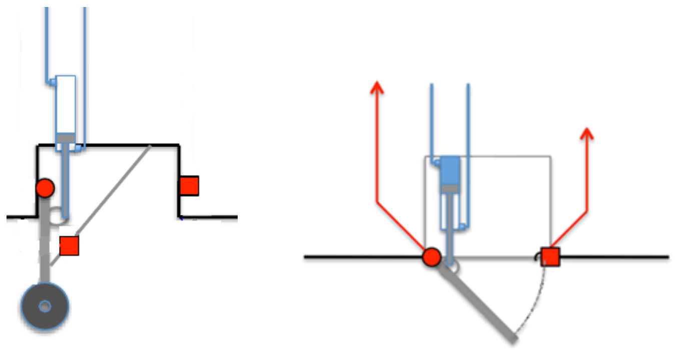


Fig. 4. Visualization of the physical environment

In addition, the visualization is subdivided into components, where each component reflects a specific refinement level of the model. A component is only displayed if the corresponding refinement level is part of the running simulation. The visualization shown in Figure 1 illustrates the last refinement step that is described in Section 2.7. In that sense the visualization is created to be extensible, for instance with new refinement levels.

The visualization can be used for different purposes. For instance, BMotion Studio is able to replay user defined traces within the visualization. This feature helped us to check whether the two basic scenarios: the outgoing sequence and the retraction sequence are realized accurately in our model. Beside analysing the two basic scenarios, we also used this feature to replay traces that lead to invariant violations found by ProB. A stepwise visualization of door behaviour while simulating the retraction sequence is demonstrated in Figure 5.

It was also used to communicate the model between the people involved in this case study.

5 Conclusion

In this paper we presented our Event-B model of the ABZ landing gear case study. By using refinement we were able to add stepwise more details to our model and make a formal development of system manageable. Even if our model does not cover the complete specification of the system, we were able to validate some "normal mode" requirements of the system. For example, our model is capable to perform the full landing gear retraction and extension sequences, both of which can be reversed at any moment in time.

Our main goal was to verify that our toolchain is able to deal with the case study. We used the following techniques to validate our model:

- Model checking
- Constraint based model checking
- Proving
- LTL model checking
- Animation based simulation
- Trace checking
- Determinism checking
- Relative deadlock checking

Initiated by the requirements of the case study, we developed a determinism checker and relative deadlock checker. We believe these new ProB features are also useful for future projects.

We used BMotion Studio to create a visualization of the model that was crucial in the development and validation of the model. Conversely, we used the case study to experiment with the development version of BMotion Studio and ProB 2.0.

Although developing a visualization required extra effort, the benefits of the visualization were tremendous. The visualization helped to get a common understanding about the model. It revealed problems and errors in the model. The arrangement of the visualization into different refinement steps allowed us to hide some of the complexity of the system and to focus on a specific problem. We strongly believe that the ability to animate and visualize the system is crucial for correctness and for reducing modelling effort when developing non-trivial formal models.

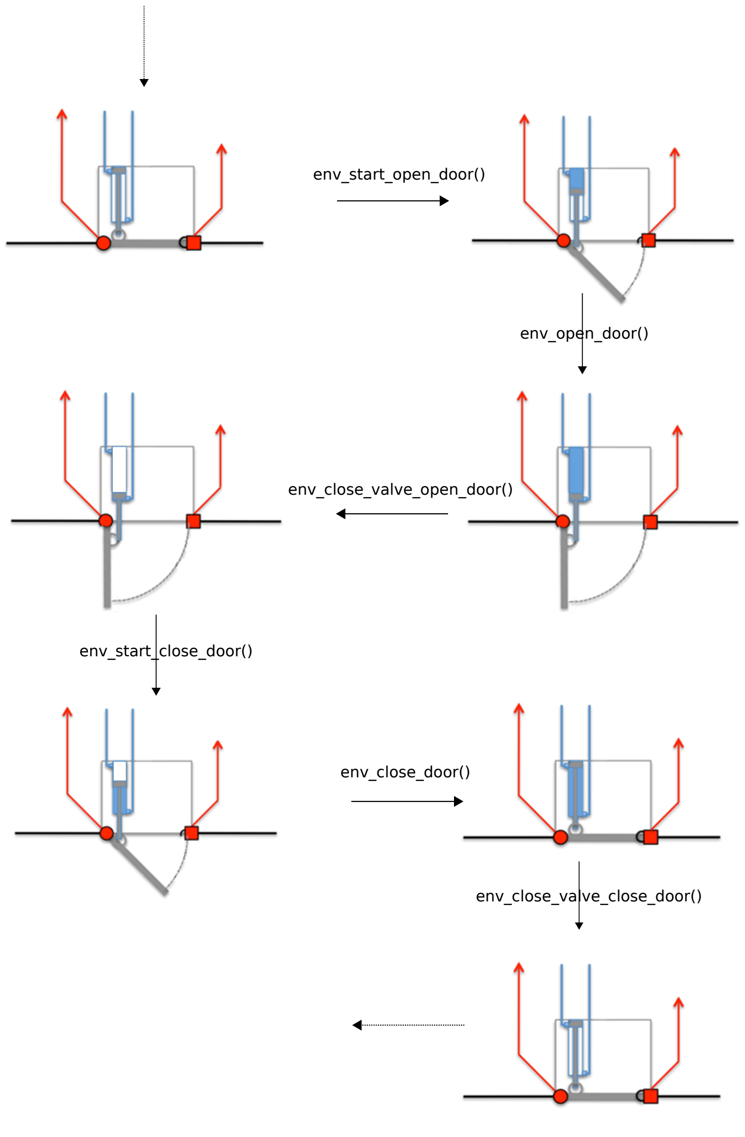


Fig. 5. Physical door behaviour while simulating retraction sequence

6 Possible Future Work

As part of the research project Advance we have developed a co-simulation framework that allows to use ProB together with another simulator for continuous systems. The framework uses the Functional Mock-up Interface (FMI) standard to exchange information between the simulators. We could use the framework to simulate the controller of the landing gear system in ProB and the environment (or parts of it) in another simulator, e.g. Dymola.

Another interesting work would be to use our visualizations for formalisations of the other case study solutions, including those that are specified in CSP, TLA$^+$ or Z. The visualization could be also enhanced with interactive components (e.g. buttons) to drive the simulation.

References

1. J.-R. Abrial. *The B-Book*. Cambridge University Press, 1996.
2. J.-R. Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010.
3. J.-R. Abrial, M. Butler, and S. Hallerstede. An open extensible tool environment for Event-B. In Z. Liu and J. He, editors, *Proceedings ICFEM'06*, LNCS 4260, pages 588–605. Springer-Verlag, 2006.
4. J. Bendisposto, M. Birkhoff, J. Clark, I. Dobrikov, M. Fontaine, F. Fritz, R. Goebbels, D. Hansen, P. Kantner, P. Koerner, S. Krings, L. Ladenberger, L. Luo, M. Leuschel, D. Plagge, and C. Spermann. ProB 2.0 source code. Available at `http://github.com/bendisposto/prob2`.
5. J. Bendisposto, M. Birkhoff, J. Clark, I. Dobrikov, M. Fontaine, F. Fritz, R. Goebbels, D. Hansen, P. Kantner, P. Koerner, S. Krings, L. Ladenberger, L. Luo, M. Leuschel, D. Plagge, and C. Spermann. ProB 2.0 Update Site for Rodin. Available at `http://nightly.cobra.cs.uni-duesseldorf.de/experimental/updatesite/`.
6. R. Gmehlich, K. Grau, S. Hallerstede, M. Leuschel, F. Lösch, and D. Plagge. On fitting a formal method into practice. In S. Qin and Z. Qiu, editors, *Proceedings ICFEM'2011*, volume 6991 of *Lecture Notes in Computer Science*, pages 195–210. Springer, 2011.
7. L. Ladenberger, J. Bendisposto, and M. Leuschel. Visualising Event-B models with B-Motion Studio. In *Proceedings FMICS'2009*, LNCS 5825, pages 202–204. Verlag, 2009.
8. M. Leuschel and M. J. Butler. ProB: an automated analysis toolset for the B method. *STTT*, 10(2):185–203, 2008.
9. R. Silva and M. Butler. Shared event composition/decomposition in event-b. In B. K. Aichernig, F. S. de Boer, and M. M. Bonsangue, editors, *FMCO*, volume 6957 of *Lecture Notes in Computer Science*, pages 122–141. Springer, 2010.