Modelling and Analysing the Landing Gear System: a Solution with Event-B/Rodin

Pascal André, Christian Attiogbé and Arnaud Lanoix

AeLoS Team
LINA CNRS UMR 6241 - University of Nantes
{firstname.lastname}@univ-nantes.fr

Abstract. This paper presents a solution to the landing gear system case study using Event-B and Rodin. We study the whole system (both the digital part and the controlled part). We use *feature augmentation* to build an abstract model of the whole system and *structural refinement* to detail more specifically the digital part. The required safety properties are formalised and proved. We propose a specific approach to deal with a family of reachability properties. The experimentations conducted during the study are supported by the Rodin tools. We show that the presented solution is systematic and it can be applied to similar case studies.

1 Introduction

This work reports on an answer to the landing gear system case study submitted to the state/proof-based formal methods community, in the scope of the ABZ conference. Our motivation is to contribute to the challenges of this real-life case study with Event-B, to share and to compare experiences and knowhow with other competitors. Even if the Event-B method have been widely used in many academical [1] and industrial case studies [2,3,4,5,1,6,7], there are still some challenging questions about methods, liveness properties management, reusable specification patterns, etc. However Event-B is recognised as a mature, well-researched formal development method which is equipped with mature engineering tools such as Atelier-B and Rodin. Accordingly we decide to present a complete solution using Event-B. By the way we have not only to answer to the case study but also to explore solutions to the challenging questions in Event-B.

We address in our proposal the two main challenges that characterise the considered case study viewed as representative of critical embedded systems. The challenges are firstly the modelling of the control part of the landing system and secondly the proof of the safety requirements. Accordingly the contribution of our work is manyfold: *i)* a complete abstract modelling of the control digital part in interaction with its physical environment using a stepwise refinement with feature augmentation; *ii)* the proof of some safety properties of the landing system; *iii)* the refinement of the digital part to take into account its composition with two redundant modules as described in the requirements. Our modelling approach is summarised in Fig. 3; the presentation of the article also follows the structure depicted in this figure.

We consider that the reader is familiar with Event-B and we do not introduce the method and its features in this document; but the interested reader can consult [1] for a detailed introduction to Event-B.

The article is structured as follows. Section 2 provides the main idea and the working hypotheses of our solution to the landing gear system. In Sect. 3 we detail how the global abstract model of the system have been built with feature augmentation and we explain how we capture the requirements of the landing system. Section 4 is devoted to the structural refinement where we provide some details for the decomposition of the global model and to the detailed specification of the digital part. In Sect. 5 we discuss our solution; we give a synthesis of the approach and how it can be reused in similar control systems. We give some feedbacks on the experimentations achieved with Rodin. Finally, section 6 concludes the article and provides some perspective works.

2 Analysing and Capturing the Requirements

In the category of critical embedded system, the landing system is essentially a control system with time constraints. In such control systems the whole system is made of a controller and a controlled physical environment which interact via sensors and actuators. In the requirement document of the landing system the controller is called the *digital part*; the controlled mechanical and hydraulic environment is called the *physical part*. We use this terminology along the article. The digital part monitors and controls the physical part; the sensors provide to the digital part the information on the state of the physical part; the actuators engage the actions of the controller on the physical part.

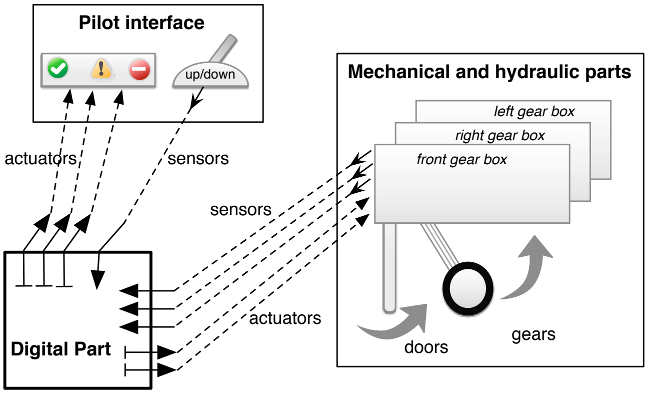

Fig. 1. Global architecture of the Landing system

2.1 Analysing the landing system requirement

The landing system requirements have already been structured in a way that there is a clear separation between the digital part and the physical part. The physical part is mainly made of gears and doors; but there is also a cockpit viewed as a control and supervisory equipment; it is made of a handle driven by a human pilot. The digital part is located between the handle and the physical part (see Fig. 1): the orders from the digital part to the controlled (physical) part are originated from actioning the handle; two main interactions are described; an interaction between the handle and the digital part and an interaction between the digital part and the controlled part.

To understand and to capture precisely the details of the components of the landing system and their behaviours, we have read several times (and continuously along the modelling work) the requirement documents and compare our views ; we have tested various approaches in the team:

- we have sketched many informal figures;
- we have drawn many state machines to capture the behaviour of doors, gears and the sequences of operations to extend or to retract the gears;
- we have built UML diagrams to capture the data and the dynamic behaviour of both the control part and the physical devices; an example of a state diagram describing the doors behaviour is depicted in Fig. 2;
- we have constructed preliminaries classical and Event-B specifications to identify data and behaviours;
- we have used Z schemas to capture data structuring at this first level of analysis.

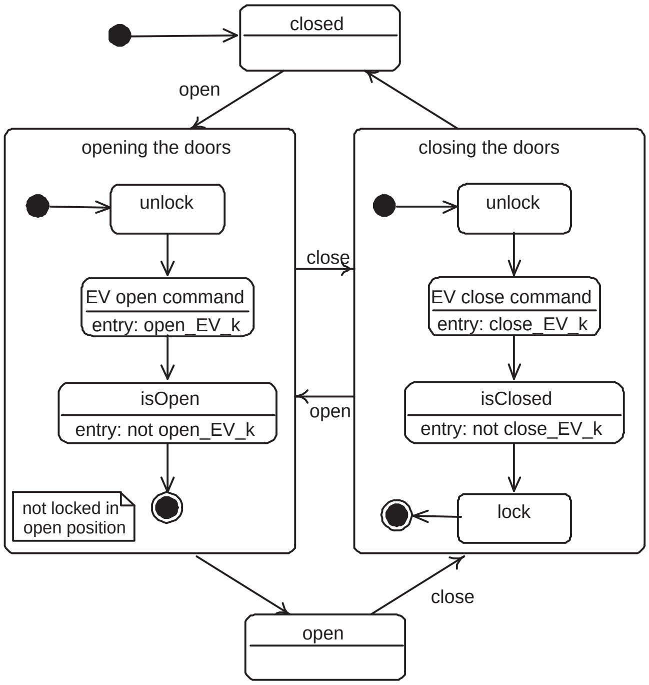

Fig. 2. The door state automaton

From the methodological point of view, it is greatly beneficial to proceed in this way; indeed some facets of the requirements are easily revealed by one approach or the other. This results in a accurate understanding of the requirement and help in building a complete abstract model. A collection of drawing informal figures, UML diagrams, Z schemas and additional works related to this step of our work can be found in a dedicated url[1].

We have classified the requirements listed in the case study document (see page 18-19 of the requirement document) in several categories of properties to be proved for the system.

Safety: The requirements R_2, R_3, R_4 and R_5 should be considered through safety properties.
Liveness: The requirements R_1 are related to liveness (reachability) properties.
Nonfunctional: The requirements R_6, R_7 and R_8 (Failure mode requirements) are related to nonfunctional properties: management of time constraints.

The physical part is made of autonomous physical devices and its global behaviour is a composition of the behaviours of the considered devices. We consider as in the requirement document that the physical devices exist and we will not build them; the challenge deals with the control part only (see page 2 of the requirement document). However we have to consider an abstraction of the physical part in order to build a global model of the landing system detailed enough to capture the required properties.

The digital part is perceived first at the abstract interface level and detailed (refined) progressively. A top-down approach as used in Event-B is appropriate.

2.2 Modelling Methodology

The landing system is studied globally by considering both the digital part and the physical part but only the digital part will be studied in details.

It has been established and demonstrated in several case studies [4,8,1] that complex systems can be constructed by combining horizontal refinement (with feature augmentation in an abstract model) and structural refinement (making the abstract structures more and more concrete).

According to the complexity of the landing case study we use feature augmentation to build the abstract model; indeed the landing system at a first approximation contains the digital part and the physical part; but each of these parts is also made of several structuring details which can be introduced step by step.

From the point of view of methodology one question is how to determine the starting point of the abstract model. There are different ways to proceed with.

i). One can start by specify the physical part with a correct behaviour and progressively adding the digital part which preserves the behaviour of the physical part.
ii). One can start with a desired digital only described by its interface with the (future) physical part, and progressively adding details on the physical part.

[1] `http://www.lina.sciences.univ-nantes.fr/aelos/softwares/LGS-ABZ2014/`

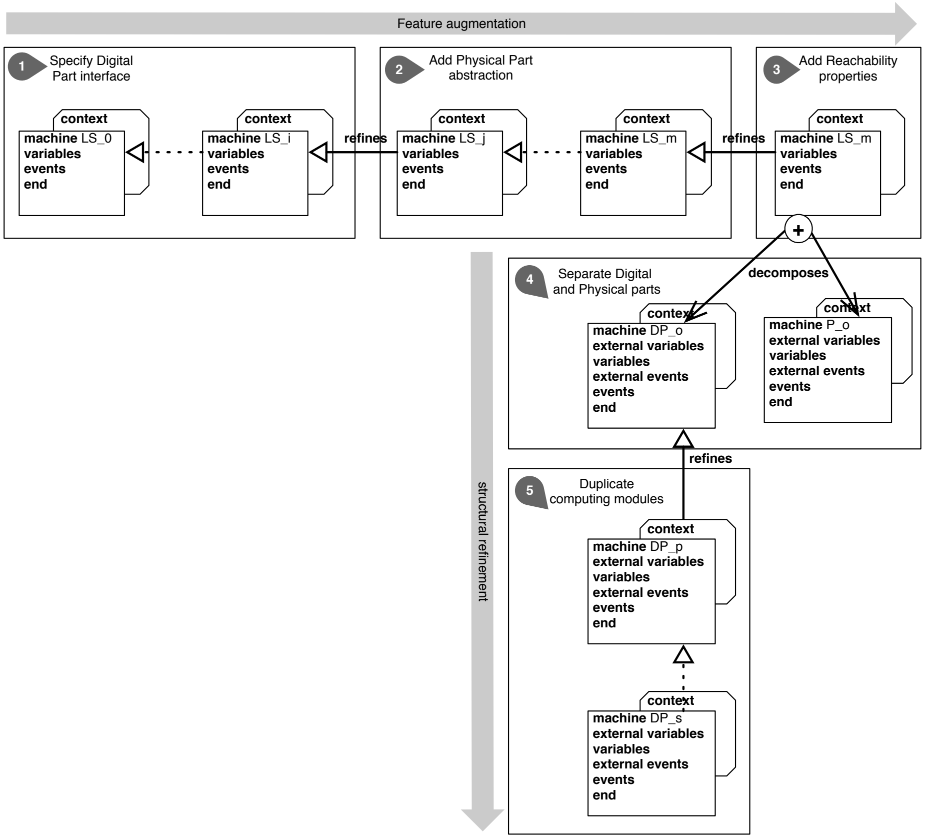

Fig. 3. Principle of the modelling approach

iii). A conjoint construction of the digital part and the physical part as soon as the details of each part are needed to express either the properties or the needed structuring details of the whole system.

We have experimented with the three approaches. It appears that the second one is more relevant: we firstly specify the digital part focusing on its interface with the physical part. Then, we have to take into account the elements of the physical part before describing some properties of the digital one (for example the reachability properties). The interaction cycle sense/decision/order of control systems requires the availability of a state space including the data of both sides.

Consequently, following the system engineering approach of Event-B, we will build an abstract model including first an abstract view of the digital and then introducing the physical part; this is achieved using a series of horizontal refinements (by feature augmentation).

We distinguish three main levels[2] of refinement:

- **Level 1 (external view of the digital part):** we focus on the interface of the digital part and the related events. Inside this level, the features will be introduced progressively (interface, actions of the outgoing sequence, triplicated sensors, etc.)
- **Level 2 (introducing physical devices with their sensors):** the digital part is progressively linked to abstractions of the physical devices which simulate their real behaviours.
- **Level 3 (introducing reachability constraints):** properties related to time and reachability are dealt with; specific constructions are introduced for this purpose.

The result of these refinements is a full abstract model of the landing system, where a lot of requirements are expressed and proved. At **level 4**, this full abstract model will be decomposed into a model of the physical part and a model of the digital part; we use the *Event-B decomposition* approach to manage this step of the process.

The **level 5 (duplicating the computing modules)** details the digital part with a series of vertical refinements (called *structural refinements*). We distinguish here two main levels of refinement.

- *Introduction of the two computing modules:* the refinement at this level enables us to model the internal structure of the digital part, the two redundant modules are introduced. This step is structured with small refinement steps where we consider one category of the interface variables at a time: the inputs and then the outputs variables.
- *Modelling of the internal behaviour of the computing module:* in the current stage of our work, the last step of the structural refinement is the modelling of the internal behaviour of each computing module.

In the remaining sections of the article we detail each level listed above.

[2] note that each level is a set of very small steps of refinement

3 Building an Abstract Model of the Landing System with Feature Augmentation

The method we used is to start with a rough abstract model with few desired properties and to incorporate little by little more details until to reach an almost complete model of the whole system comprising both the digital part and the controlled physical part.

3.1 Stepwise construction of the abstract model

The requirement document is helpful (see page 7 of the document) to identify the different variables at the interface between the digital part and the physical part (Fig. 4); they are the input and output variables introduced in pages 6 and 7 of the requirement document.

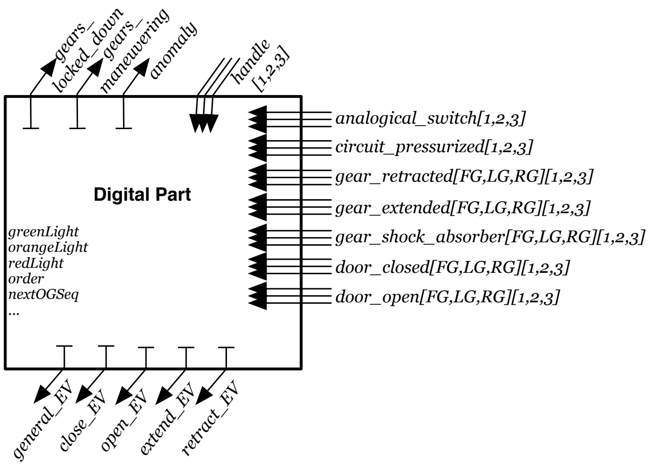

Fig. 4. The interface of the digital part

Therefore, we distinguish three categories of variables at the interface of the digital part.

- the *input* variables: *handle*, *gear_extended*, *gear_retracted*, *analogical_switch*, *door_closed*, *door_open*, etc.,
- the order *output* variables: *general_EV*, *close_EV*, *open_EV*, *extend_EV* and *retract_EV* which are used to send orders to the physical part and
- the state *output* variables: *gear_locked_down*, *gears_manoeuvring* and *anomaly* which are used to interact with the pilot interface.

We have identified the following basic enumerated sets:

- $HSTATE = \{hDown, hUp\}$, which denotes the state of the *handle* variable;

- $AnalSWSTATE = \{openSW, closedSW\}$ which is used to model the *analogical_switch* variable;
- $DOOR = \{FD, RD, LD\}$ which denotes the front, the right and the left doors;
- $GEAR = \{FG, LG, RG\}$ which denotes the front, the right and the left gears.

The input variables are triplicated, as indicated in the requirement document. They are modelled with a type $TRIPLE = \{1,2,3\}$ used as an index of the function variables; their descriptions are as follows:

$$\boxed{\begin{array}{l} handle \in TRIPLE \to HSTATE \\ analogical\_switch \in TRIPLE \to AnalSWSTATE \\ gear\_extended \in (TRIPLE \times GEAR) \to BOOL \\ gear\_retracted \in (TRIPLE \times GEAR) \to BOOL \\ door\_closed \in (TRIPLE \times DOOR) \to BOOL \\ door\_open \in (TRIPLE \times DOOR) \to BOOL \\ \ldots \end{array}}$$

For instance with the function variable $handle \in TRIPLE \to HSTATE$ we capture precisely that $handle_i \in \{hDown, hUp\}$ with $i \in \{1,2,3\}$.

The state output variables are modelled as follows:

$$\boxed{\begin{array}{l} gears\_locked\_down \in BOOL \\ gears\_maneuvering \in BOOL \\ anomaly \in BOOL \\ \ldots \end{array}}$$

The order output variables are modelled as follows:

$$\boxed{\begin{array}{l} general\_EV \in BOOL \\ close\_EV \in BOOL \\ open\_EV \in BOOL \\ \ldots \end{array}}$$

Additionally to these three categories of the interface variables, we have some *internal* variables; they are used inside the controller. For example, to manage the state of the lights which indicate the position of the gears and doors to the pilot, we have *greenLight*, *orangeLight*, *redLight*. They are bound to the output state variable *gears_locked_down* with an invariant predicate. In the following section we have other example of such variables.

Handling normal mode and anomaly Two modes are distinguished: a *normal* mode where the system is controlled digitally and its behaviour is correct if there is no anomaly detected in the system otherwise a permanent failure is observed; an *emergency* mode where the system is controlled analogically. Only the normal mode is in the scope of the case study (see page 1 of the requirement document).

Accordingly the boolean state output variable *anomaly* is used to denote that an anomaly has been detected or not (either $anomaly = TRUE$ or $anomaly = FALSE$).

This variable is used to raise an anomaly and also to distinguish the both modes in the specification of properties and in the event guards.

One of the challenges of this case study is first the construction of an abstract model which is as faithful as possible with the requirements. One of the properties of the landing system is *at the same time doors cannot be closed and open*. This is captured by the following predicate into the invariant of the building model:

$$\boxed{anomaly = FALSE \Rightarrow ran(door\_closed) \cap ran(door\_open) = \varnothing}$$

Deriving events from the action sequences The digital part of the system controls the hydraulic devices according to the pilot orders and also to the mechanical devices positions (page 13 of the requirement document). This is achieved according to two specific action sequences: a landing action sequence and a retraction action sequence. A internal variable $order \in HSTATE$ is used to pass on the order issued by actioning the handle. The variable *order* is an example of *internal* variable.

Moreover we have to take into account the requirements R_2 and R_3 (page 18 of the requirement document) which give the conditions of the orders (UP or DOWN) that can initiate the landing and retraction sequences and the conditions to enable them.

A thorough analysis of the two action sequences (outgoing sequence and retraction sequence) of the landing system helps us to capture the behaviour of the digital part. Even if they are nested each sequence is analysed precisely; it is made of a sequence of transition from state to state; each sequence is started as the effect of an action on the handle by the pilot. The remaining transitions in the sequence are mainly the orders from the digital part to the physical part, provided that some conditions described in the requirement document are established; therefore we are able to build a set of events (with conditions-actions) that describe as Event-B guarded events, the outgoing sequence or the retraction sequence.

In order to control perfectly the evolution of the outgoing sequence we use a variable *nextOGseq* which indicates in the event guards the next step in the outgoing sequence. The variable is updated in the body of the events.

Modelling the outgoing sequence We now describe more precisely the outgoing sequence to illustrate our approach.

This outgoing sequence is defined at the page 14 of the requirement document. It starts with the order DOWN and is finished when the gears are extended and door closed. Between these events we have an interaction involving the digital part (which issue the orders), the physical parts which execute the orders and the sensors which provides the various states of the gears and doors. Figure 5 depicts the global behaviour of the outgoing sequence, we can distinguish the main events and how they interact with the other components which are modelled: sensors and actuators. The labels of the transitions in the figure are the events that model the behaviour of the digital part with respect to the outgoing sequence. Note that the guards of the events depend on the current state of the transition system and on the state of the digital part (the sensors). In Fig. 5 we use the brackets to indicate the elements of the guards. The second sequence —retraction sequence— is modelled in the same way.

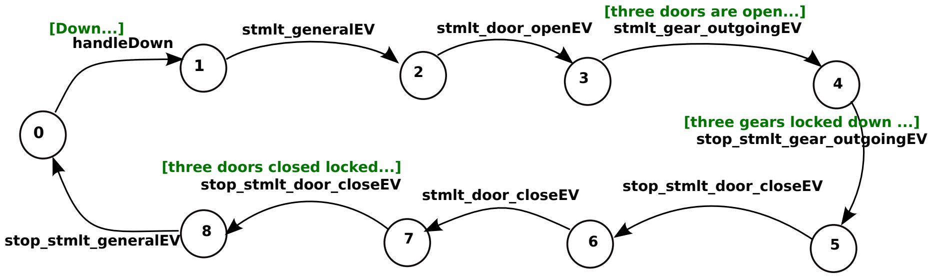

Fig. 5. The synthesis of the outgoing sequence

From the methodological point of view, we are still in the **level 1** of Fig. 3; several horizontal refinements, depicted by the dashed arrow in the figure, are necessary to integrate gradually the variables and the related events.

Method of the construction of the events The starting point is the state space obtained with the three categories of variables at the interface of the digital part and the fourth category of the internal variables. We define a family of events related to each category of the interface variables: *monitoring events*, *control events*, *sensing events*. Note that the construction of the abstract model is achieve

Monitoring events family The state output variables are used for monitoring; to set these variables according to the current state of the controller and the input data, we define for each variable (for example *gear_locked_down*) of this category an event named after the variable (*monitor_gear_locked_down*) with the prefix *monitor_*. The events of this family modify the appropriate state output variables. An example of these events is the following one where the state output variable *gears_locked_down*, is modified, according to the current state and the input variable *gear_extended*.

```
event monitor_gears_locked_Down
/* page 7 : the outputs are synthesised by each module
   from sensors data and from the situation awareness
page 15 : gear_locked_down = true iff the 3 gears are seen as locked
   in extended position */
where
    @g1  ran(gear_extended) = {TRUE} // the 3 gears are seen as locked
    @gano anomaly = FALSE // no anomaly detected
then
    @a1  gears_locked_down := TRUE
    @a2  greenLight := lightON
    @a3  orangeLight := lightOFF
    @a4  redLight := lightOFF
end
```

Control events family The order output variables are used to specify the actions that stimulate the physical devices; for this purpose, each variable of this category is set with an event named after the variable with the prefix *stmlt_*, for example *stmlt_general_EV, stmlt_extend_EV*, etc. These events use the internal variables, the input variables and the output variables. An example of these events is the following one where the order output variable *extend_EV* is modified.

```
event stmlt_gear_outgoing
/* stimulate gear outgoing electro valve
   ** action 3 ** of outgoing sequence
   once the three doors are in the open position */
where
   @g0  general_EV = TRUE
   @g1  order = hDown
   @g2  ran(handle) = {hDown}
   @g3  ran(door_closed) = {FALSE} // the three doors are in the open position
   @g4  ran(door_open) = {TRUE}
   @next  nextOGseq = 3
   @gano  anomaly = FALSE// no anomaly detected
   @notretract  retract_EV = FALSE
then
   @a1  extend_EV := TRUE
   @a2  nextOGseq := nextOGseq + sequenceStep  // action 4 or action 2
end
```

Associated with these events to stimulate the physical devices, we have as many events to stop the stimulation of the devices. These events have their name prefixed with stop_ *e.g.* stop_stmlt_gear_outgoing sets the variable *extend_EV* to *FALSE*.

Sensing events family The input variables are read by the digital part but they are set by the sensors which are outside the digital part. For these variables we define the events named after the variables with the prefix *sense_ e.g.sense_gear, sense_door*, ... At this step of feature augmentation, these events *anticipate* their real future specifications, which are related to the physical part introduced latter.

An example of these events is

```
event sense_gear // anticipated
where
   @noAno  anomaly = FALSE
then
   @a0  gear_extended :∈ (TRIPLE × GEAR) → BOOL
end
```

Note that these events are completed with specific variables and events used to detect anomaly, to detect the failure of physical devices, etc.

Properties integrated in the abstract model The properties to be proved (requirements given in pages 18-19 of the requirement document) are formalised as first order predicates integrated into the invariant of the abstract model and proved along the horizontal refinement. Most of the normal mode requirements are safety properties. Requirement R_1 needs a specific treatment presented in the sequel.

- Requirements R_{21} and R_{22}.

R_{21}	We can not observe a retraction sequence (consequence of the order hUp) if the handle is down. Using the enumerated set $HSTATE$ which permits only one value from two for the variable *order*,
$order = hDown \Rightarrow ran(handle) \neq \{hUp\}$	

R_{22}	In a similar way we cannot observe an outgoing sequence (consequence of the order *hDown*) if the handle is up.
$order = hUp \Rightarrow ran(handle) \neq \{hDown\}$	

- Requirements R_{31} and R_{32}.

R_{31}	The gears outgoing event occurs if doors are open locked
$(extend\_EV = TRUE \Rightarrow ran(door\_open) = \{TRUE\})$	

R_{32}	The gears retraction event occurs if doors are open locked
$(retract\_EV = TRUE \Rightarrow ran(door\_open) = \{TRUE\})$	

- Requirement R_{41} and R_{42}.

R_{41}	Opening and closing doors electro-valve are not stimulated simultaneously
$\neg(open\_EV = TRUE \wedge close\_EV = TRUE)$	

R_{42}	Outgoing and retraction gears electro-valve are not stimulated simultaneously
$\neg(extend\_EV = TRUE \wedge retract\_EV = TRUE)$	

- Requirement R_5.

R_{51}	It is not possible to stimulate the manoeuvring EV (opening, closure, outgoing or retraction) without stimulating the general EV
$((open\_EV = TRUE \vee close\_EV = TRUE$ $\vee extend\_EV = TRUE \vee retract\_EV = TRUE)$ $\Rightarrow general\_EV = TRUE)$	

3.2 Capturing the behaviours of the physical part

One more step of the refinement by feature augmentation is the addition of the behaviour of physical devices: the sensors, the doors, the gears, as illustrated by Fig. 6. Several refinement steps are necessary to integrate gradually the variables and events related to the physical devices; this corresponds to the dashed line in the **level 2** of Fig. 3.

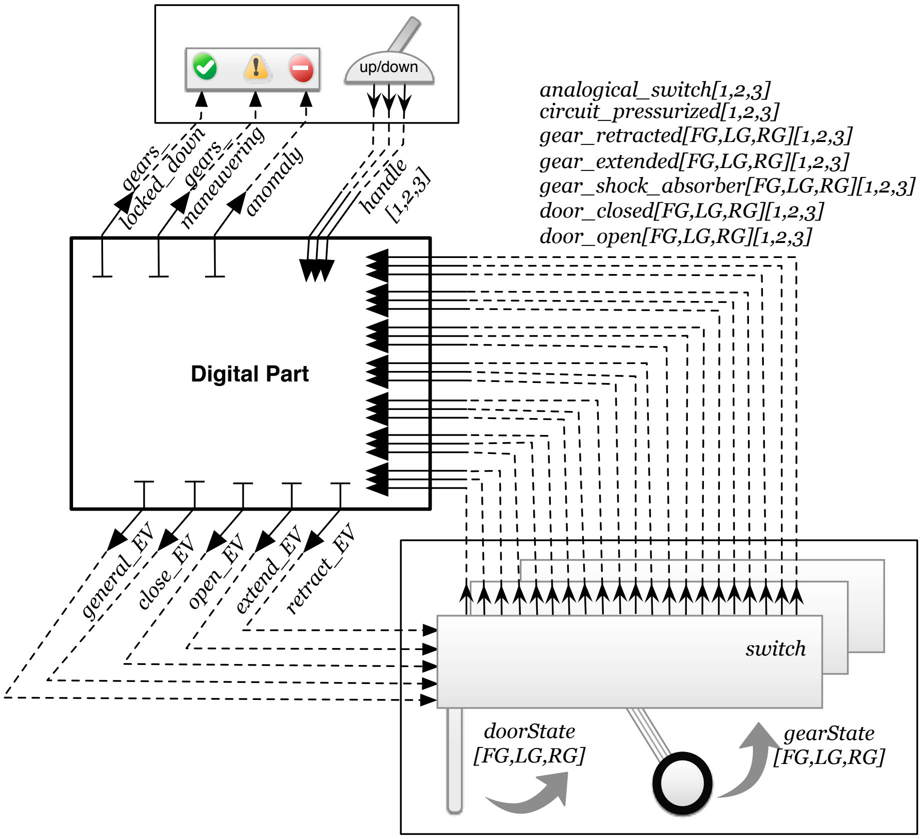

Fig. 6. The digital part connected to the physical environment

Door behaviour The door behaviour is described at page 11 of the requirement document. It is first captured with a state automata; the transitions of the automata are then described as events. For this purpose we use a transition function $doorState \in DOOR \to DSTATE$ where $DSTATE = \{ClosedLocked, ClosedUnlocked, OpenUnlocked\}$ is the enumerated set of the identified door states Fig. 2. The set $DOOR$ contains the three doors. The function $doorState$ is a total function; this captures the requirement that all the three doors are controlled via the state transition.

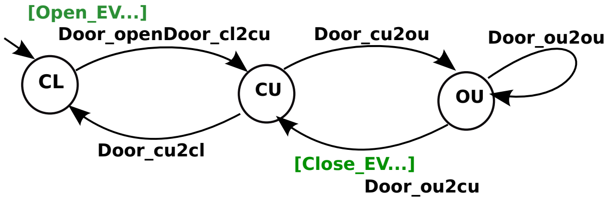

Fig. 7. Behaviour of doors

The starting transition of the door behaviour is enabled by the $open\_EV$ order[3] given by the digital part. Therefore there is a synchronisation between the digital part and the doors. The remaining transitions are handled with events which are conveniently guarded in such a way that we have the correct ordering of the doors. Note that the three doors are simultaneously controlled by the orders issued by the digital part.

```
event Door_openDoor_cl2cu
/* the three doors
Door's Behaviour
first transition of the Door automata
when the action open_EV is performed by the control system */
where
    $@g1$  $open\_EV = TRUE$ // all doors EV are on
    $@g2$  $ran(doorState) = \{notOpenLocked\}$
then
    $@a1doorState := DOOR \times \{notOpenNotLocked\}$ // door is being opened
end
```

In the same way the transition starting from the open to close position is synchronised with the $close\_EV$ order given by the digital part. Therefore we have the complete interaction between between the orders from the digital part and the doors.

Gear behaviour The gear behaviour is specified in the same way as the doors. A specific transition function is used: $gearState \in GEAR \to GSTATE$ where $GSTATE =$

[3] In Fig. 7 the brackets indicate the event that contributes to enable the transition.

$\{RetractedLocked, RetractedUnlocked, ExtendedUnlocked, ExtendedLocked\}$ is an enumerated set of gear states. The labels of the transition correspond to the events that model the behaviour of the gears. The set *GEAR* contains the three gears.

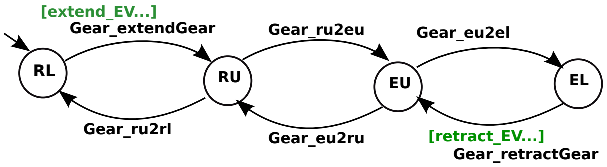

Fig. 8. Behaviour of the gears

Sensor behaviour The sensors behave like the observers of the door states and the gear states. When a door reaches the open state, then the related sensor events set the variable *door_open* to *TRUE*.

The related events in the specification are prefixed with *sense_*. The modelling of the sensors follows faithfully the requirement document; that is each sensor is made of three micro-sensors; each of them has its own state. We illustrate the modelling with the following event which deals with the extension of one micro-sensor[4] of the front gear; the event sets the corresponding value in the interface variable (*gear_extended*).

```
event sense1_FGE_OK // is sensor1 of Front Gear extended ?
refines sense_gear
any nge
where
    @noAno  anomaly = FALSE
    @g1  gearState(FG) = LockE // the Front Gear is seen Locked Extended
    @g2  nge ∈ (TRIPLE × GEAR) → BOOL
    @g3  nge = gear_extended <+{(1 ↦ FG) ↦ TRUE} // update of the var
then
    @ea2  gear_extended := nge // that is (1 ↦ FG) := TRUE
end
end
```

Note that a sensor can have an abnormal functioning. To simulate this anomaly on the sensor, the corresponding event could set the state of the gear to the wrong value. For this purpose, we define in our model such events, named likely sense1_FGE_KO, which set a wrong value.

[4] identified by 1

3.3 Handling some reachability requirements

Based on the idea of Lamport's logical clocks [9], we implement a technique that captures the reachability requirement R_1 given in page 13 of the requirement document. For that purpose, we introduce the notion of *control cycle*. A *control cycle* is a period of time during which one can observe several events, especially a chain of events denoting an outgoing sequence or a retraction sequence; a typical control cycle is one starting with an event which denotes the *hDown* order and terminating by an event which denotes the fact that "*the gears are locked down and the doors are seen closed*"; similarly, another control cycle is started when the handle triggers an order hUp. A dedicated variable $endCycle$ is used to control the start and the end of each control cycle.

Assume that we have observable events that occur along the time and that denote our events of interest[5]; for instance the starting of an outgoing sequence, a door closed, a gear locked in a position, etc. Each such event can be stamped with the timestamp of its occurrence, thus if we have the set of observed events we can define at least a partial ordering of these events (see Fig. 9).

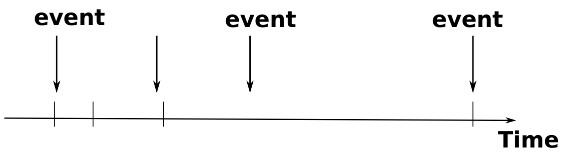

Fig. 9. Events and timestamps

Given a set $obsEvents$ of events and a logical clock modelled as a natural number, the occurrences of the events can be ordered by the timestamp given by the clock. In our case two events cannot happen at the same time. We use a partial function $ldate \in obsEvents \pfun \mathbb{N}$ to record the timestamps of the events. We can compare and reason on the timestamps of any events happening during a sequence and specifically within the specific event sequence called *control cycle*.

For example, in the normal mode, we observe the event "*the door is closed and the gear extended*" (named dcge) at the end of a cycle, if the event "*order DOWN is given*" (named downH) occurs and is maintained (no event upH occurs). If these events have respectively the specific timestamps dj and di, then we can compare di and dj and also examine the events which happen between di and dj. Accordingly the property R_{1bis} of the requirement is expressed as follows:

$$\boxed{\begin{array}{l}\forall dj.(((dj \in \mathbb{N}) \land (dcge \in dom(ldate)) \land (dj = ldate(dcge)) \\ \quad\quad \land (endCycle = TRUE) \land dj < llc) \Rightarrow \\ \quad \exists di.((dd \in \mathbb{N}) \land (downH \in dom(ldate)) \land (di = ldate(downH)) \land (di < dj) \land \\ \quad\quad\quad \forall ii.(ii \in \mathbb{N} \land di \leq ii \land ii < dj \Rightarrow ldate \sim [\{ii\}] \neq \{upH\})))\end{array}}$$

[5] These events are not to be confused with Event-B events.

The above property means that if we reach the end of a control cycle where the door is closed and the gear extended at a given timestamp (dj), then we should have an order *hDown* issued at a timestamp di less that dj and maintained between dj and di; the outgoing sequence is not interrupted by an order hUp which would start another cycle. Consequently we have expressed the property R_{11bis}. Property R_{12bis} can be expressed in a similar manner.

To put in practice in Event-B with Rodin, we defined the set $obsEvents$ in the context of our machines, and the above property is included in the invariant of the abstract model.

This step of the horizontal refinement process is the **level 3** of Fig. 3.

3.4 Well-ordering the control of the physical part

The modelled behaviour of the landing system especially the model of the digital part should attend the two main objectives given at pages 13 of the requirement document: *"to control the physical devices"* and *"to monitor the system and inform the pilot"*.

Control systems have essentially three main steps: sensing of input variables (set by external mechanisms), making a decision (compute some values with respect to the current state), actuating the controlled mechanisms (by modifying their values via the actuators). This constitutes the elementary interaction cycle sense/decision/order which is repeatedly applied to control a system.

The behaviour described by our model should be conform to the interaction cycle. It is the case; each control cycle initiated by the pilot via the handle, to achieve an outgoing sequence or a retraction sequence, is made of several interaction cycles sense/decision/order. Indeed, in the normal mode of the landing system, the outgoing sequence and the retraction sequence structure the first objective of controlling the physical device; a control cycle is performed according to each sequence. As explained in Sect.3.1 (page 9) our model is structured according to the two sequences; each sequence is made of a chain of events which guards are defined in such a way that[6] the control implements the interaction cycle sense/decision/order which involve the events of the three families identified (please, see page 10). Note that the sense step involves the events of the *sensing* family; the decision step involves the events of the *monitoring* family; the order step involves the events of the *control* family. Moreover the events of the *monitoring* family achieve the second objective of the landing system.

4 Building the Digital Part with Structural Refinement

Structural refinement starts when we have finished the construction of the abstract model of our system integrating the digital part, the gear boxes and the pilot interface, and we have proved all the necessary requirements.

During the structural refinement only the digital part will be refined with the objective to build the software system. Consequently we concentrate the refinement on the variables and events of the digital part. The variables and events which are specific

[6] they follow the state transition

to the behaviour of the physical part are not refined but we keep them in the model in order to preserve *animation capabilities*. This approach is very pragmatic. In addition to the proofs, the animation provides a concrete view of the system behaviour and this helps in gaining confidence in the model and also in a validation process with a client as example. Therefore the model is not decomposed as we have presented in Sect. 2: the model will be decomposed into two models at the end of the structural refinement, as depicted in Fig. 10.

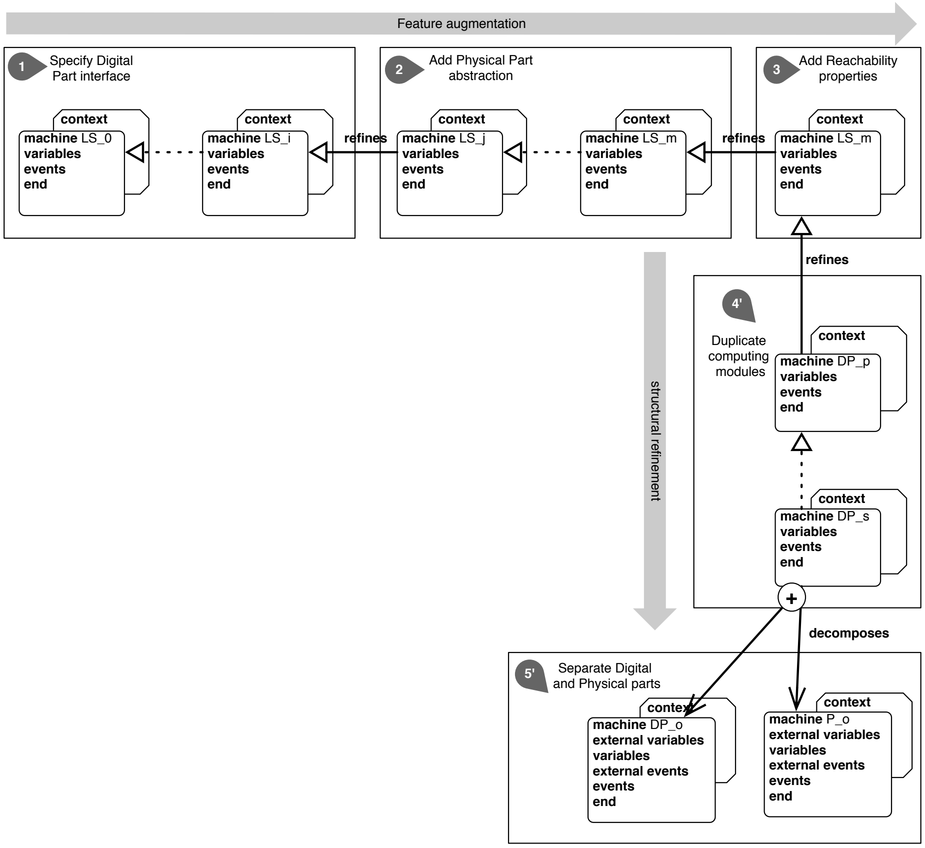

Fig. 10. Following modelling approach (a variant of Fig.3)

In the following, we explain how we refine the digital part; the starting point is the global abstract model obtained from the previous section (Sect. 3).

4.1 Refinement of the digital part

The requirement document details the inner structure of the digital part, which is made of two redundant computing modules (see the Digital architecture depicted in Fig.5 of the requirement document). The policies to compose the inputs and the outputs of the two modules are explained (pages 5-6 of the requirement document). Some structural refinements are helpful to capture faithfully this architecture of the digital part.

Introducing the two computing modules with a refinement The two modules have the same interface (input and output variables) inherited from the abstract model of the digital part.

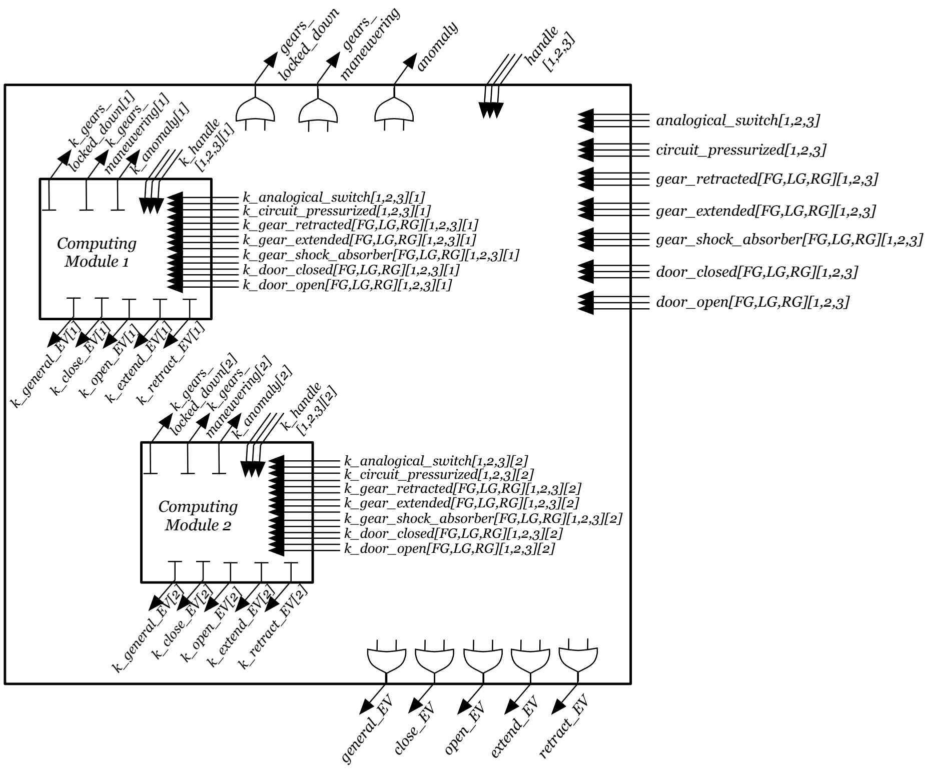
gears_ locked_down
gears_ maneuvering
anomaly
handle [1,2,3]
analogical_switch[1,2,3]
circuit_pressurized[1,2,3]
gear_retracted[FG,LG,RG][1,2,3]
gear_extended[FG,LG,RG][1,2,3]
gear_shock_absorber[FG,LG,RG][1,2,3]
door_closed[FG,LG,RG][1,2,3]
door_open[FG,LG,RG][1,2,3]
k_gears_ locked_down[1]
k_gears_ maneuvering[1]
k_anomaly[1]
k_handle [1,2,3][1]
Computing Module 1
k_analogical_switch[1,2,3][1]
k_circuit_pressurized[1,2,3][1]
k_gear_retracted[FG,LG,RG][1,2,3][1]
k_gear_extended[FG,LG,RG][1,2,3][1]
k_gear_shock_absorber[FG,LG,RG][1,2,3][1]
k_door_closed[FG,LG,RG][1,2,3][1]
k_door_open[FG,LG,RG][1,2,3][1]
k_general_EV[1]
k_close_EV[1]
k_open_EV[1]
k_extend_EV[1]
k_retract_EV[1]
k_gears_ locked_down[2]
k_gears_ maneuvering[2]
k_anomaly[2]
k_handle [1,2,3][2]
Computing Module 2
k_analogical_switch[1,2,3][2]
k_circuit_pressurized[1,2,3][2]
k_gear_retracted[FG,LG,RG][1,2,3][2]
k_gear_extended[FG,LG,RG][1,2,3][2]
k_gear_shock_absorber[FG,LG,RG][1,2,3][2]
k_door_closed[FG,LG,RG][1,2,3][2]
k_door_open[FG,LG,RG][1,2,3][2]
k_general_EV[2]
k_close_EV[2]
k_open_EV[2]
k_extend_EV[2]
k_retract_EV[2]
general_EV
close_EV
open_EV
extend_EV
retract_EV

Fig. 11. Principle for adding the two modules

As far as the method is concerned, Figure 11 depicts the principle of our solution. Each interface variable of a module k (where $k \in \{1,2\}$) is inherited from a variable (for instance *gear_extended*) of the digital part of the abstract model and it is denoted by $k\_gear\_extended(k)$ where (k) is an index. The prefix *k_* of the variable name enables us to keep the same name but to differentiate the abstract variable name and the refined one. An enumerated set $CompModule = \{1,2\}$ is used for the indexes. Therefore each interface variable of the computing modules is specified with the following shape, where the abstract variables are indexed.

$k\_handle \in CompModules \rightarrow (TRIPLE \rightarrow HSTATE)$
$k\_gear\_extended \in CompModule \rightarrow ((TRIPLE \times GEAR) \rightarrow BOOL)$
$k\_gear\_retracted \in CompModule \rightarrow (TRIPLE \times GEAR) \rightarrow BOOL$
$k\_analogical\_switch \in CompModule \rightarrow (TRIPLE \rightarrow AnalSWSTATE)$
$k\_door\_closed \in CompModule \rightarrow (TRIPLE \times DOOR) \rightarrow BOOL$
$k\_door\_open \in CompModule \rightarrow (TRIPLE \times DOOR) \rightarrow BOOL$
$\ldots$
$k\_general\_EV \in CompModule \rightarrow BOOL$
$k\_extend\_EV \in CompModules \rightarrow BOOL$
$k\_close\_EV \in CompModule \rightarrow BOOL$
$k\_open\_EV \in CompModule \rightarrow BOOL$
$\ldots$

Notice, at this stage, the interface of the two modules are not linked with that of the abstract module. This is achieved via further refinement features: the binding between the variables and their k-indexed forms.

Spawning the inputs inside the computing modules We have to specify that the inputs of the digital part (either from the cockpit or from the sensors) are the same ones for the computing modules. The principle adopted here is that the value of each input variable (for example *handle*) at the abstract level is pushed in the corresponding variable (for example *k_handle*) of each computing module. As the inputs of the modules are (and should be) the same, an invariant is defined in each case of variable spawning in order to guarantee the correctness of the binding between the input variable of the digital part and the same input of the computing modules.

$handle \in ran(k\_handle)$ /* binding invariant */
$gear\_extended \in ran(k\_gear\_extended)$ /* binding invariant */

We use the following event pattern to spawn the variables at the interfaces of the computing modules.

```
event spawn_handleDown // spawn handleDown within the k CompModules
where
    @g1  $ran(handle) = \{hDown\}$
then
    @a1  $k\_handle := \{1 \mapsto (TRIPLE \times (ran(handle))),$
             $2 \mapsto (TRIPLE \times (ran(handle)))\}$
end
```

Consequently the identified specification rule is that a new event is introduced along with each new k-indexed variable. This event should copy the variable at high level (the digital part) into the indexed variables at the low level. This specification rule is reusable in all such cases where variables from a module should be buried inside submodules.

Furthermore, the existing events which guards or actions involve spawned variables should be refined by extending their guards and actions in order to satisfy the binding between the variables and the associated k-indexed variables.

As far as the input from the sensors are concerned, specific treatment is needed in order to take account of the separate behaviours of the three micro-sensors that can send different values event wrong values. For this purpose, considering (front, left, right) doors or gears, for each one two events are needed to specify the impact of the micro-sensor behaviour on the k-indexed input variables of the computing modules; one event gives the right value of the micro-sensor the other event gives the wrong value.

One noticeable feature in this case is that when we have a nondeterministic event of abstract level (as for the value of the sensors), then in the refinement the event should be refined (not extended).

as an example, we consider the sensor 2 from the three sensors that sense if the left door is open (*LDO*), hence the event names *sense*2_*LDO*_*OK*, *sense*2_*LDO*_*KO*.

```
event event sense2_LDO_OK // seonsor2 of LDoor
extends sense2_LDO_OK
any nkdo
where  $@noano\ \ \forall kk\Delta(kk \in CompModules \Rightarrow (k\_anomaly(kk) = FALSE))$
    $@dnkge\ \ nkdo \in CompModules \to ((TRIPLE \times DOOR) \to BOOL)$
    $@nge\ \ nkdo\ \ = CompModules \times \{door\_open < +\{(2 \mapsto LD) \mapsto TRUE\}\}$
then
    $@ea1k\_door\_open := nkdo$
end
```

```
event event sense2_LDO_KO // sensor2 of FDoor (simulating malfunctioning)
extends sense2_LDO_KO
any nkdo
 where
    $@noano\ \ \forall kk\Delta(kk \in CompModules \Rightarrow (k\_anomaly(kk) = FALSE))$
    $@dnkge\ \ nkdo \in CompModules \to ((TRIPLE \times DOOR) \to BOOL)$
    $@nge\ \ nkdo = CompModules \times \{door\_open < +\{(2 \mapsto LD) \mapsto FALSE\}\}$
 then
    $@ea1\ \ k\_door\_open := nkdo$
end
```

Merging the outputs of the computing modules The specification principle is as follows. As presented in the requirement and depicted in Fig. 11, the k-indexed output variables (for example $k\_extend\_EV(1)$ and $k\_extend\_EV(2)$) are merged using a logical OR to set the corresponding variable (for example $extend\_EV$) at the output of the digital part. Therefore the event that sets the variable should be guarded by the availability of the merged value. As explained before, a binding invariant should be provided for each variable and the related k-indexed variable.

```
$k\_extend\_EV \in CompModules \to BOOL$
$extend\_EV \in ran(k\_extend\_EV)$ /* binding invariant */
```

The merging is illustrated with the following event *merge_stmlt_gear_outgoing*, which updates the output variable *extend_EV*. Only the guard should be extended in the refinement since the abstract event already has the right substitution to set the variable.

```
event merge_stmlt_gear_outgoing
/* MERGING the result of the k module to stimulate the gear_outgoing
    ** action 3 ** of outgoing sequence */
extends stmlt_gear_outgoing
    any kk
    where
    @gkk  kk ∈ CompModules
    @theOR  k_extend_EV(kk) = TRUE /* OR : one of them is TRUE */
end
```

We identified this specification as a **promotion pattern** which promotes the outputs of the modules at the level of the digital part. This pattern is reusable in all cases of modules encapsulation.

As for the orders to the physical part, the output variables for the cockpit result from the merging of the related output variables of the computing module; therefore the event that modify these variables are refined by extension of their guards. This is illustrated with the following event.

```
event merge_monitor_gears_locked_Down
/* page 7 : the outputs are synthesised by each module
from sensors data and from the situation awareness ... */
extends monitor_gears_locked_Down
    any kk
    where
    @gkk  kk ∈ CompModules
    @gr1  k_gears_locked_down(kk) = TRUE /* OR : meaning one is true */
end
```

Specifying the behaviour of the computing modules The two computing modules have the same behaviour. This behaviour is the one sketched with the three categories of variables at the interface of the digital part. Typically we have the events that monitor the system and set the state output variables, the events that give orders to the physical part and the impact of the input variables on the state of the digital part.

The principle here is to define as for the variables, the k-indexed form of the events related to the three categories of the interface variables and the internal variables.

```
event k_stmlt_general_EV
/* anticipated, will be refined inside the k=1,2 modules */
any nkge
where
    $@dkge\ \ nkge \in CompModules \to BOOL$
    $@vnke\ \ nkge = CompModules \times \{general\_EV\}$
then
    $@a1\ \ k\_general\_EV := nkge$
end
```

4.2 Decomposition into a digital part and a physical part

A decomposition paradigm is supported by the Event-B method. Two approaches exist[7] for this purpose: the Abrial'style decomposition (called the A-style decomposition) [10] based on shared variables, and the Butler'style decomposition (called the B-style decomposition) [11,12] based on shared events. In the A-style decomposition, events are first split between Event-B sub-components and then shared variables of the sub-components are used to introduce external events in the sub-component; these external events should be refined in the same way. In the B-style decomposition, variables are first partitioned between the sub-components and then shared events (which use the variables of both sub-components) are split between the sub-components according to the used variables.

We have used the A-style decomposition which is more relevant when considering the events that describe the behaviour of two different parts of the landing system. It is straightforward to list the events that describe the behaviour of the physical part in order to separate them from the events closely related to the digital part. For this purpose we have experimented the decomposition plugins of the Rodin toolkit using the A-Style decomposition approach.

As shown in Fig. 10, the global abstract model resulting from the horizontal refinement (**Level 3**, Fig. 10) is refined vertically (**Level 4'**, Fig. 10). The resulting refined model should be decomposed into a digital part and a physical part (**Level 5'**, Fig. 10).

Consequently, the digital part must be separated from the environment, i.e. the physical part and the pilot interface. The methodological guide to achieve the decomposition is as follows:

- the digital part is made with the events defined in two families of events (see Sect.3.1) related to the interface variables: *monitoring events* and *control events*.
  - The events in the monitoring family are all those with the names prefixed by *monitor_*. Examples are: monitor_gears_locked_Down, monitor_gears_maneuvering, monitor_anomaly.
  - The events in the control events family are those with the names prefixed with *stmlt_* and *stop_stmlt_*. Examples are stmlt_general_EV, stmlt_door_Opening, stmlt_gear_outgoing, stop_stmlt_general_EV, stop_stmlt_door_opening, stop_stmlt_gear_outgoing.

[7] implemented in the Deploy Project.

- all the events of the last family (*sensing events*) are in the physical part.

As a matter of fact, it is possible to define a systematic process to guide the decomposition process.

5 Discussion: Coverage of the Requirements and Assessment

We begin the section with the analysis of the experimental results; then we discuss the coverage of our study and experimentation. Methodological shortcomings are commented and we finish by generalising our approach to help the interested readers to reuse our work in similar case studies.

5.1 Experimentation with Rodin and statistics

Rodin The Rodin tool is very efficient for proving the Event-B models; a very high percentage ($\sim$ 90%) of proof obligations was automatically discharged. Note that the current version of the Event-B models is partial as we focus on representative events instead of being exhaustive specifications. The specifications are available on a website[8].

	Total	Auto	Manual	Reviewed	Undisch.
LandingSys5	619	547	6	0	66
Abstract model					
Landing_DP_Ctx	0	0	0	0	0
LandingSysDP_A	115	114	1	0	0
LandingSysDP_SWITCH_A	5	3	0	0	2
LandingSysDP_DOOR_A	42	42	0	0	0
LandingSysDP_DOOR_GEAR_A	79	79	0	0	0
LandingSysDP_DOOR_GEAR_TIME_A	2	2	0	0	0
Models of the vertical refinement					
LandingSysDP_DGT_R1_In	52	50	0	0	2
LandingSysDP_DGT_R2_INOUT	56	56	0	0	0
LandingSysDP_DGT_R3_INOUTDOOR	128	81	5	0	42
LandingSysDP_DGT_R3INOUTDOORGEAR	140	120	0	0	20

Table 1. Statistics of PO generated and proved with Rodin

Statistics on Proof Obligations are given in Tab. 1. From a total of 619 POs, 547 of them were automatically discharged by Rodin and 6 of them were interactively discharged. Most of the POs at the abstract levels were proved. The undischarged POs are related to the structural refinement and specifically they are related to the binding invariants.

[8] http://www.lina.sciences.univ-nantes.fr/aelos/softwares/LGS-ABZ2014/

However the Rodin tool is inefficient with large models; it lacks of space and becomes very slow and unpractical. Therefore managing very large models requires a rigorous slicing and several small steps of refinements. This is the reason why we have introduced many refinements, but it is still not enough, the slicing should be more fine.

As far as the ProB animation tool (integrated in Rodin) is concerned, it is very helpful to tune the Event-B models; but unfortunately for the large models as the ones built (several finite sets, several variables and several events) for the landing system the animation tool is inefficient. It lacks of space and results in frozen work space. This is cumbersome as unfortunately we was not able to animate the last steps of our development. In a similar way, the ProB model checker was only able to exploit the preliminary abstract models; but as soon as we introduce more complex state space and events, we were not able to model check the models.

5.2 Coverage and assessment

The proposed Event-B specification presented in this article covers the main aspects of the landing system: the digital part with modules redundancy, its physical part (mechanical and hydraulic environment and pilot interface) and their interactions.

We have emphasised a treatment of the global features of the landing system. Therefore we have dealt with all the aspects of the control of the physical part, starting from an handle action on the pilot interface. Only the *outgoing* sequence is treated here. The second sequence should be treated in a similar way. Each step of the interaction between the digital part are treated: we have a cycle to sense the environment, to make a decision, to give an order. This cycle is repeatedly observed.

The table 2 provides a synthesis of the coverage of the requirements. The presented work covers mainly the safety properties; liveness properties are treated by adapting Lamport's logical clocks [9]; nevertheless we have not deal with time constraints. In our study of the landing system we have considered representative properties. Indeed the requirement document lists at pages 18 and 19, two categories of properties: normal mode and failure mode requirements. But in each category the stated requirements are quite similar.

Category	Covered requirements	Uncovered requirement
Safety	R_2, R_3, R_4 and R_5	
Liveness	R_1	R_{1bis}
Nonfunctional		R_6, R_7 and R_8

Table 2. Coverage of the requirements

Code generation was out of our solution. We build on the experiments of several case studies where the Event-B was used and where some methodological guidelines was provided [8,4]. Accordingly, feature augmentation and structural refinement methods appear very efficient.

5.3 Methodological shortcoming

One flaw of the Event-B top-down approach is the constraint imposed by the evolution of the global abstract model defined before its refinement to the concrete models. For example in Fig. 12, M_{30} stands for the global abstract model; only the models of the first horizontal line (indexed by 10,20,...) and the last column (indexed by 3O) are pertinent during the specification (the models that are inside the box are not considered. This constraint prevents for an incremental model evolution. Indeed, if we miss some features in the abstract state, we will have to reconsider completely the structural refinements. It would be interesting to be able to mix both horizontal and vertical refinements in an incremental view of the design method as shown by the intermediary steps inside the box of Fig. 12. In [13] Back, have proposed guidelines for this purpose; an adaptation of this work to Event-B is likely to be interesting.

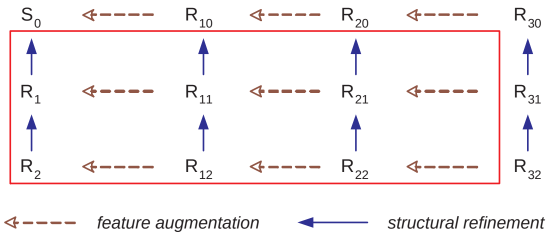

Fig. 12. Combining horizontal and vertical refinements

The reuse of existing independent models, with a bottom-up approach, would be interesting for managing large Event-B models. A typical example is the composition of existing models to build a given abstract model where each part can be modified and refined separately.

5.4 A Generic approach for a control system

Many features of our solution emphasise the genericity of the used approach. During our work on the landing system, which is viewed as representative of critical embedded control systems, we pay attention to emphasise the features that can be reused in similar case studies.

Roughly, Fig. 13 depicts a general principle that may governs the organisation of event-based models of control systems. We consider that the high level state space of the control system can be described on the basis of the elicitation of the interface variables between the control part and the physical part of the considered systems.

The dashed ovals are representative of the parametric events; they are linked to the both family of events (sensing family or control family). They should be replaced by the effective events related to the logic of a specific case study. For example in the case of the landing system the merging event is a logical OR; in a different case study the merging may correspond to another specific policy.

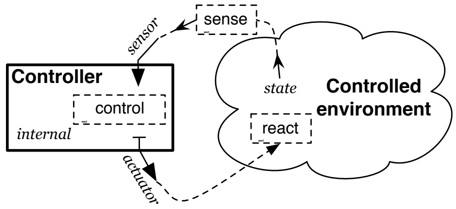

Fig. 13. Generic Event-B based control System

The case where the internal control modules (sub-controllers) have the same interface as the main control module (the controller) is a specific case; it is structured by the meta-model in Fig. 14.

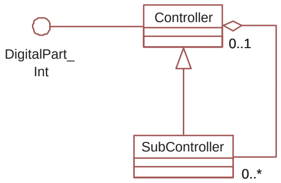

Fig. 14. Incremental refinement pattern of duplicated controllers

Additional specific features and modelling guidelines are the following.

i). *Description of the interface variables of the controller:* the interface variables into three categories (input variables, output variables and internal variables); each category of variables are set by the events which are classified in both categories: sensing, or controlling. These event families are conform with the sense/decision/-control control cycle.

ii). Use of *feature augmentation to bind the controlled environment:* this is achieved on the basis of the sensing events, which in turn need the description of the controlled environment. Abrial's advices on proceeding with small steps of refinements are very helpful here. Instead of a big step of refinement including several variables and events, several small steps of refinements dedicated to variables and events, in an incremental way, are more efficient.

iii). *Reachability property with partial ordering:* specific events (not exactly at the same granularity with the B model events) with timestamps are systematically used to order and to reason on reachability properties.

iv). Use of *Structural refinements* based on the control events to refine the controller. The involved categories of variables are the internal variables; the input variables are spawned inside submodules if any; in the same way output variables should be updated by promotion from the submodules if any.

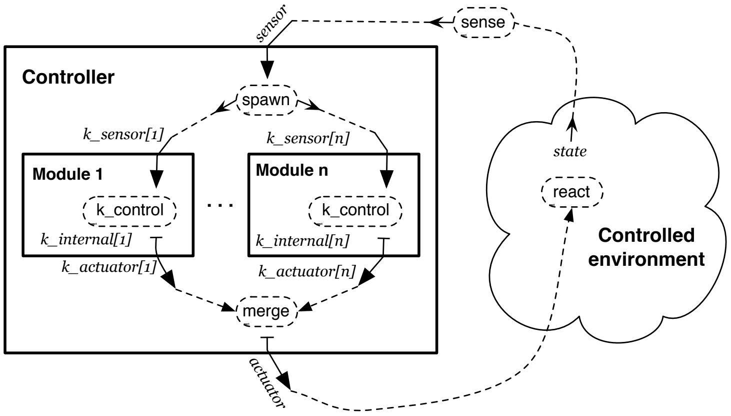

Fig. 15. Generic Event-B based control system with modules redundance

v). *Composition of several redundant blocks:* when a controller is made of several redundant modules, it is straightforward to describe a generic module and use an indexing function to compose several instances of such modules (see Fig. 15).

- *Immersing variables inside modules:* the values coming from outside one or several modules can be systematically immersed inside the modules with an event such as *spawn_gear_extended*.
- *Promoting variables outside a module:* in a symmetric way, the values going outside a module or several modules are systematically described using a promotion pattern described in Sect. 4.1 for merging the output variables of the internal computing modules.

When the modules are not redundant, each one should be refined conveniently, but the described treatment of the inputs and outputs is the same.

Figure 16 shows Event-B patterns at each —horizontal and vertical— refinement level, from the most abstract model ControlSystem0 describing only the controller interface to the systematic decomposition into two parts: Environment2 and Controller3.

6 Conclusion

We presented a complete study of the landing gear system from the point of view of the modelling and the verification of given properties. It is a contribution to the specification challenge submitted to the ABZ communities. We used Event-B and the related horizontal and vertical refinement approaches. An important part of the requirement document has been treated by considering both the digital part and the physical part. The digital part has been refined until its decomposition into the two redundant modules introduced in the requirement. Code generation is not attacked.

After the requirement capture where we used several approaches, we proceed by a systematic approach which can be reused in similar studies of reactive embedded system. Describing precisely the interface between the digital part and its environment

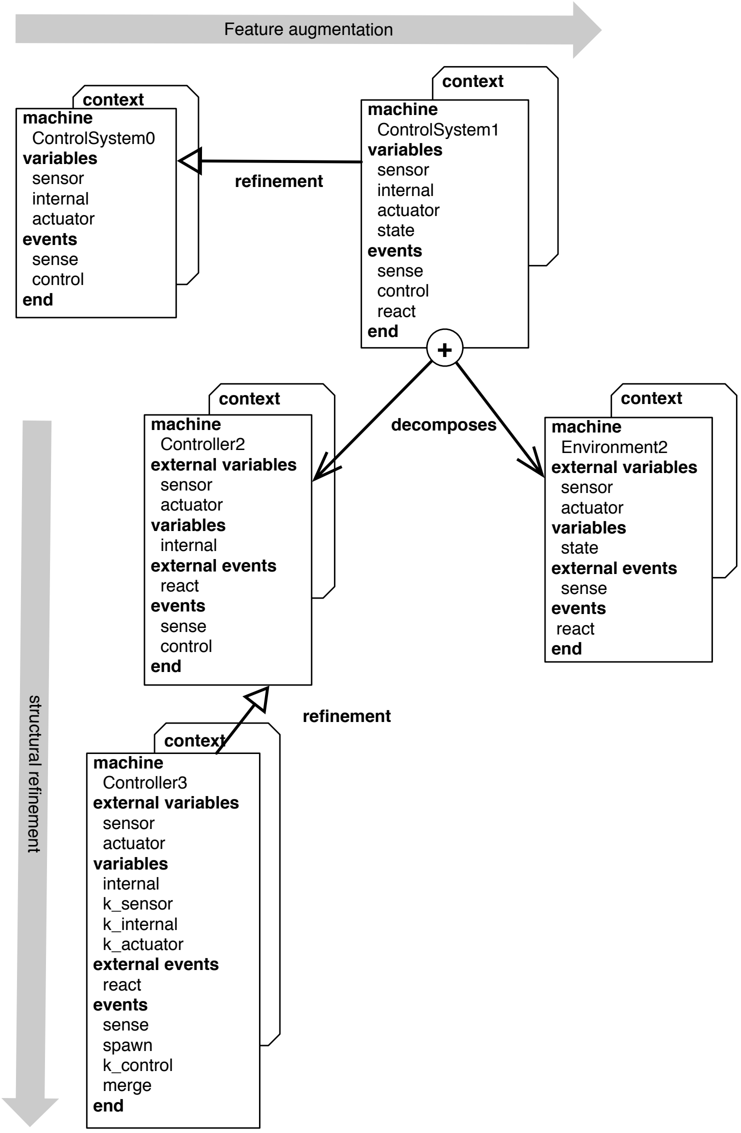

Fig. 16. Generic Event-B models

(pilot interface, physical devices) is an important starting point to build the abstract model of the control system. The variables defining the interface are systematically classified into four categories which structure the model of the control system and its further refinements. We have defined some event description patterns related to the four variable categories which structure the control system. Families of events are identified to structure the modelling of the interaction between the digital part and its environment. The events needed to build and refine the digital part depends on the interface variables and they are systematically described by considering the standard sense/decision/order control cycle. It appears that each family of events is precisely located either in the horizontal refinement or in the vertical refinement. Consequently our approach is systematic; it can be reused with the provided guidelines and it can be assisted by tools. Moreover, we have proposed a new approach to deal with reachability properties; its is based on Lamport's logical clocks [9] and the partial ordering of specific events introduced in addition to the Event-B events.

Many lessons was drawn from our experimentation. The use of Event-B is relevant to attack the landing system. Indeed few properties related to time constraints and reachability are not deal with. We have not spent time to extend the method to overcome this flaw. But we have shown that, with some crafty modelling approaches we can come up with a part of these reachability properties. This opens the way for improving the Event-B method and enriching its tools.

We plan to generalise the generic approach presented here and build a related tool to help in modelling and structuring the development of embedded control systems. The proposed approach muse be compared to the four relational variables model proposed by Parnas [14] for rigorous system development. The approach we have used to deal with reachability properties will be studied in more details and extended to tackle more properties.

References

1. Abrial, J.R.: Modeling in Event-B - System and Software Engineering. Cambridge University Press (2010)
2. Butler, M., Yadav, D.: An Incremental Development of the Mondex System in Event-B. Formal Asp. Comput. **20**(1) (2008) 61–77
3. Lanoix, A.: Event-B Specification of a Situated Multi-Agent System: Study of a Platoon of Vehicles. In: 2nd IFIP/IEEE International Symposium on Theoretical Aspects of Software Engineering (TASE 2008), France (June 2008) 8 pages
4. Damchoom, K., Butler, M.J.: Applying Event and Machine Decomposition to a Flash-Based Filestore in Event-B. In Oliveira, M.V.M., Woodcock, J., eds.: SBMF. Volume 5902 of Lecture Notes in Computer Science., Springer (2009) 134–152
5. Cansell, D., Méry, D., Proch, C.: System-on-chip design by proof-based refinement. STTT **11**(3) (2009) 217–238
6. Fathabadi, A.S., Rezazadeh, A., Butler, M.: Applying Atomicity and Model Decomposition to a Space Craft System in Event-B. In: NASA Formal Methods. (2011) 328–342
7. Méry, D., Singh, N.K.: Formal Specification of Medical Systems by Proof-Based Refinement. ACM Trans. Embedded Comput. Syst. **12**(1) (2013) 15
8. Damchoom, K., Butler, M.J., Abrial, J.R.: Modelling and Proof of a Tree-Structured File System in Event-B and Rodin. In Liu, S., Maibaum, T.S.E., Araki, K., eds.: ICFEM. Volume 5256 of Lecture Notes in Computer Science., Springer (2008) 25–44

9. Lamport, L.: Time, Clocks, and the Ordering of Events in a Distributed System. Commun. ACM **21**(7) (1978) 558–565
10. Abrial, J.R., Hallerstede, S.: Refinement, Decomposition, and Instantiation of Discrete Models: Application to Event-B. Fundam. Inform. **77**(1-2) (2007) 1–28
11. Butler, M.: Decomposition Structures for Event-B. In Leuschel, M., Wehrheim, H., eds.: IFM. Volume 5423 of Lecture Notes in Computer Science., Springer (2009) 20–38
12. Silva, R., Butler, M.: Shared Event Composition/Decomposition in Event-B. In Aichernig, B.K., de Boer, F.S., Bonsangue, M.M., eds.: FMCO. Volume 6957 of Lecture Notes in Computer Science., Springer (2010) 122–141
13. Back, R.J.: Software Construction by Stepwise Feature Introduction. In Bert, D., Bowen, J.P., Henson, M.C., Robinson, K., eds.: ZB. Volume 2272 of Lecture Notes in Computer Science., Springer (2002) 162–183
14. Parnas, D.L., Madey, J.: Functional Documents for Computer Systems. Sci. Comput. Program. **25**(1) (1995) 41–61