A tool-assisted method for the systematic construction of critical embedded systems using Event-B

Pascal Andre, Christian Attiogbé, Arnaud Lanoix

Computer Science and Information Systems 17(1):315–338 https://doi.org/10.2298/CSIS190501042A

A Tool-assisted Method for the Systematic Construction of Critical Embedded Systems using Event-B

Pascal André, Christian Attiogbé and Arnaud Lanoix

LS2N CNRS UMR 6004 - University of Nantes
{firstname.lastname}@univ-nantes.fr

Abstract. Embedded control systems combine digital and physical components, leading to complex interactions and even complexity of their development. In [4] we proposed a method to build such complex systems in a systematic way. The overall method starts from an abstract model of the physical environment of the considered system and its controller. The method consists in a sequence of refinement steps, in the spirit of Event-B, that gradually introduces design details from an abstract level, until more concrete levels. Two main refinement processes are distinguished: one to capture the global model, the other to detail it; we provide through the method the guidelines to accompany these two refinement processes. But there were a lack of assistance tools. The designers need to be assisted by tools to guide them, to automate partially the refinements and to help in proving more easily model properties. We illustrate the method with the landing gear system case study and choosing the Event-B tool Rodin for illustration; we make it explicit the tools requirements for such a general method and, we introduce a tool support to assist the user in applying the method in combination with standard Event-B tool such as Rodin.
Key words: Embedded control systems; Modelling method; Event-B patterns; Tool

1. Introduction

Engineering complex embedded control systems requires methods and assistance tools. Without dedicated methods their analysis is painful, inefficient and time-consuming. More specifically guidelines and tools are required for formal software engineering. Unlike many other types of software, embedded systems are often developed for specific target environments (processors, vehicles, medical devices, etc.) and very often they should run for long times (even years), once they have been implemented in their so-called critical environments. Therefore, embedded systems and their construction have stringent robustness requirements; accordingly one have to develop them with the sake of reliability at runtime. There are numerous models for embedded real-time systems [9]; moreover the target environments of each embedded system do not help the construction or the expansion of dedicated tools and methods. There are many works dealing with semi-formal models extraction from requirements documents, for example in the scope of the UML notations[10]. There are also many works around the tabular requirement engineering method by Parnas[13]; but we are not aware of any results about the synthesis of Event-B formal models from informal requirments as we are doing in our work.

Considering that *i)* the requirements for reliability and correct construction of the models and the derived embedded systems are important concerns, and that *ii)* the de-

velopment of these systems still lacks of methods to guide the developers, we are motivated to contribute to fill the gap between these needs and the state of the art. Many researchers underline the need of methods, techniques and tools to support formal modeling and more especially for the Event-B approach which addresses the full software development process: [8,3,12,23]. We have proposed in [4] a correct-by-construction method (named Heñcher) dedicated to the construction of critical embedded control systems. This method, based on Event-B, is intended to guide step by step the specifier or the engineer to drive its development from requirements to concrete software, defining abstract models, and refining them in a systematic way. The current article is an extension of that previous work. We extend that work in two main directions: *i)* we extend the proposed method with an assistance tool dedicated to help the users to apply the Heñcher method step by step and to build quickly the preliminary Event-B models of her/his control systems. The tool is designed as a companion tool of the already existing Event-B frameworks such as Rodin or Atelier B for which we provide input models; *ii)* the Event-B models of the case study are now totally proved using the Rodin tool.

We present in this article the main components and the background of the tool; it is designed as an extensible standalone tool that should be compatible and integrable with Event-B framework. The article is completely reshaped compared to [4] where more space was given to the method and less to its illustration. Here we emphasise the application of the method on the Landing Gear System case study.

The article is structured as follows. In Section 2 we introduce the proposed method through its main steps. In Section 3 we present a benchmark case study, the *Landing Gear System*, we illustrate the detailed application of our method on this case study, and we emphasise issues on tool requirements. Section 4 is devoted to the proposed assistance tool through its main components and the provided facilities; and its use to illustrate parts of the case study. In Section 5 we draws some conclusions of this work.

2. A Glimpse of the Heñcher Method

In [4], we presented a stepwise and systematic method (named Heñcher) to construct critical embedded control systems using Event-B. Complex systems can be constructed by combining (see [6,7,1]) two classical approaches: 1) horizontal refinement with *feature augmentation* where we have to build a global abstract model of a the whole system (a controller and its physical environment) and 2) *structural refinement* (making the abstract structures more and more concrete).

The high-level state space of any control system can be described by the **elicitation of the interface variables** between the digital part (the controller) and the physical part (the controlled environment) of the considered system. Fig. 1 depicts a general principle that may govern the organisation of *event-based models* of control systems. The dashed ovals are representative of the parametric events families; They should be replaced by the effective events related to the logic of a specific case study.

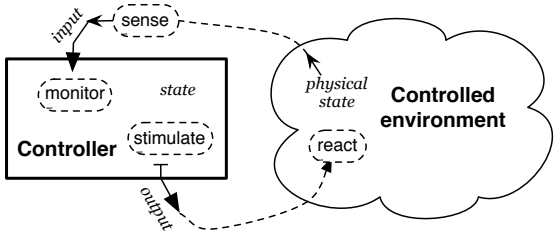

Fig. 1. A generic shape for event-based model of a control system

Besides, the identified physical devices to be controlled should be precisely listed. The behaviour of each one will be specified later.

We summarise here the Heñcher method (detailed in [4]); it starts from this interface and comprises six steps for guiding the modelling and analysis of the target system. These steps spawn the two classical Event-B structuring approaches. The horizontal process is made of 3 steps and the vertical process is made of at least 2 steps (steps 4 and 5).

Step 1: Characterising the abstract model of a considered system.

- *Step 1.1: Elicit the controller interface* made of three categories of variables describing the controlled environment (input from sensors, output to actuators, and state for monitoring). Additional *internal* variables represent information inside the controller.
- *Step 1.2: Elicit the global properties of the system*: system properties, including safety, liveness and non-functional properties.
- *Step 1.3: Start with a first abstract model* and build a first Event-B abstract model.
- *Step 1.4: List the events of the abstract model*: *sense events*, *monitor events* and *stimulate events*. These families of events are in compliance with the standard sense-decision-control of any control system's cycle. Additionally the behaviour of the physical part is described with the *reaction events family*.

Step 2: Extension of the previous abstract model using *feature augmentation* [6,7,1] to integrate the controlled environment on the basis of the *sense events* family.

- *Step 2.1:* Introduce the physical environment and the *reaction events* family.
- *Step 2.2:* Detail the *sense events* family.

Step 3: Integration of the specific properties. Considering the requirements of the system, additional specific properties are added to the global model to constrain the functioning of the system. They may be are *reachability properties* or *non-functional properties*.

Step 4: Structural refinement of the global abstract model. New internal B events may be added to refine the events of each family of events (*sense*, *monitor*, or *stimulate*). The state space variables may be refined with more details in the invariant.

The model is more refined with the behaviour of the physical part (made of the controlled devices); this is captured through the *reaction events* family.

Step 5: Decomposition into software and physical parts. We adopt the A-style decomposition [2]. The methodological guide to achieve the decomposition is as follows: the digital part is made with all the events defined in the *sense events*, the *monitor events* and the *stimulate events* families whereas the physical environment gathers all the events defined in the *reaction events* families. Each part must have an abstract view of the other.

Step 6: Refinement of the control software and the physical environment. In addition to classical but complete refinements in Event-B, we propose some guidelines to assist the user in applying there refinements steps.

- *Step 6.1:* Refining the control software. *Structural refinements* based on the *monitor* events family should be used to refine the controller. The involved categories of variables are the *input* variables, the *state* variables and the *output* variables.
- *Step 6.2:* Refining the controlled (physical) environment. Many cases can be considered depending on the system to be studied; either the physical devices are already available, or one has to build the physical devices from the formal models, or one has to build a part of the physical devices.

Proposed modelling patterns When there are sub-modules, the *input* variables may be spawned inside the sub-modules.

In the same way *output* variables may be updated by promotion from the sub-modules if any. Therefore one have to incorporate successively in the Event-B model the events to set and modify the *output* variables; they describe the result of the behaviour of the control part. State automata help to catch these behaviours; then the events of the B models encode the automata.

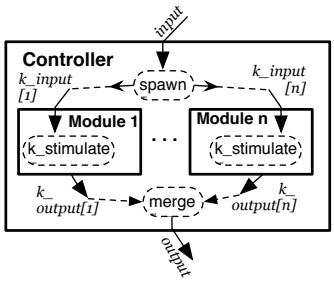

Fig. 2. Modules redundancy

We give now some recurrent patterns to help in modelling the control part.

i). *Composition of several redundant sub-modules:* 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. 2).
  – *Encasing variables inside modules:* the values coming from outside one or several modules can be systematically encased inside the modules with a dedicated event that spawn the events.
  – *Promoting variables outside a module:* in a symmetric way, the values going outside a module or several modules can be systematically described using a **promotion pattern** (with a dedicated event) for merging the output variables of the internal computing modules.

ii). When the modules are not redundant, each one should be refined separately, but the treatment we have described for the inputs and outputs variables is the same.

Fig. 3 illustrates the Event-B patterns from the most abstract model (which describes only the interface of the controller) to the systematic decomposition into two parts: the Controller and the physical Environment.

3. Applying the Method to the Running Case Study

The proposed method is applied on the *Landing Gear* (LG) case study, a benchmarking example proposed at the ABZ'2014 conference to compare different formal methods in terms of expressivity, performance, and ease of use. in [5].

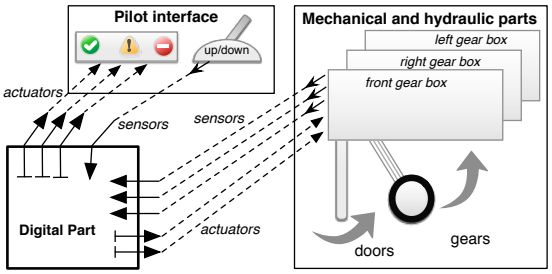

Fig. 4. Global architecture of the LG system

A prerequisite for reading this section is the detailed specification of this critical embedded system given A summary of the LG system is depicted in Fig. 4. The LG system is in charge of manoeuvring 3 landing boxes: front, left and right. Each landing box contains a landing gear, an associated door and the corresponding hydraulic cylinders in charge to move gears and doors.

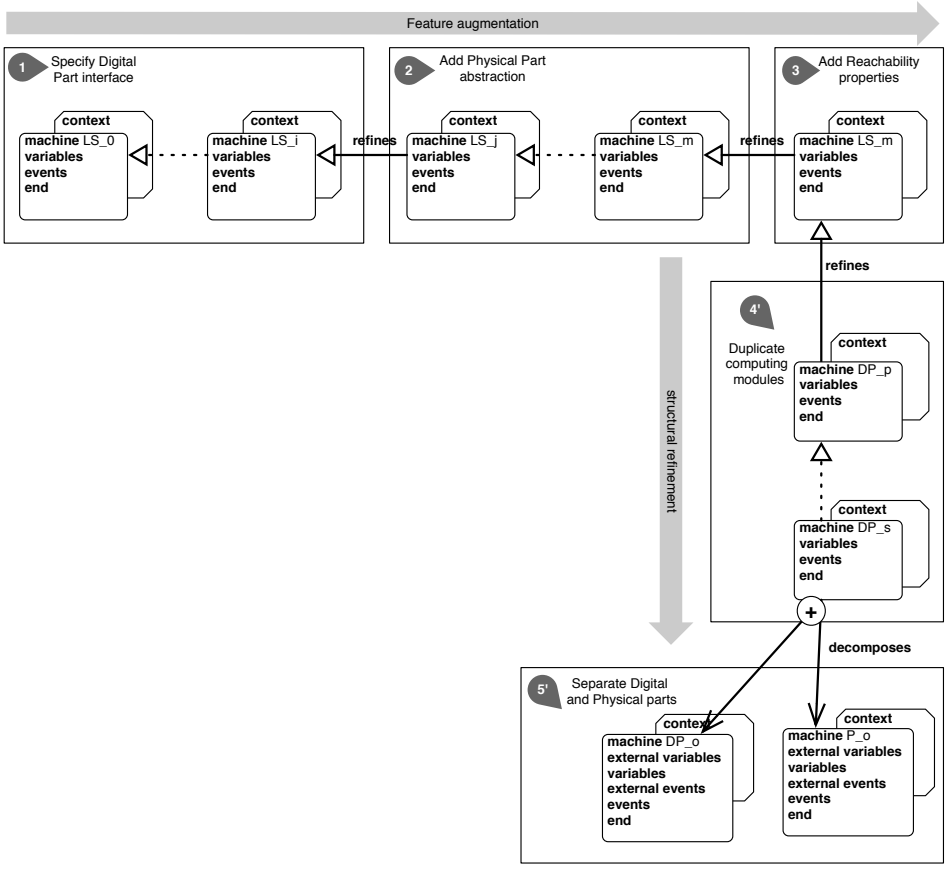

Fig. 3. Synoptic structure of the Heñcher method

The system is made of a controller (the *digital part*) and the controlled physical environment (i.e. the 3 landing gear boxes and a pilot interface) which interact via sensors and actuators.

The sensors provide to the digital part the information on the state of its physical part; the actuators engage the orders of the controller on the physical part. The physical devices already exist, we will not build them; the challenge deals with the digital control part only (see page 2 of [5]). We give the main elements resulting from the successive application of the steps proposed in the method (Sect. 2).

3.1. Horizontal Process: Building an Abstract Global Model of the System

The document [5] is helpful to identify the different variables at the interface between the digital part and the physical part.

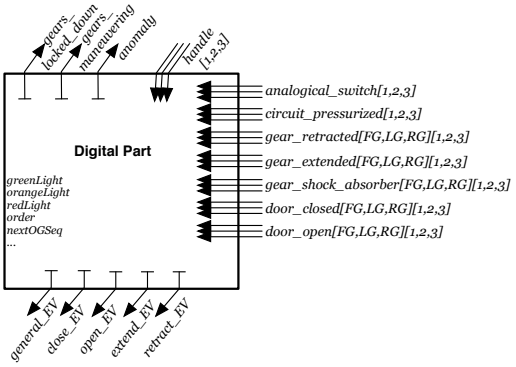

Fig. 5. The interface of the digital part

Step 1: Characterising the abstract model

Step 1.1: Elicitation and modelling of the interface variables The requirement document listed several triplicated *input* variables: handle, analogical switch, gear states, doors states$\cdots$
We model them with a type $TRIPLE = \{1,2,3\}$ used as an index of the function variables (see *Step 1.3*):

$GEAR = \{FG, LG, RG\}$	$analogical\_switch \in TRIPLE \rightarrow AnalSWSTATE$
$DOOR = \{FD, RD, LD\}$	$handle \in TRIPLE \rightarrow HSTATE$
$HSTATE = \{hDown, hUp\}$	$gear\_extended \in (TRIPLE \times GEAR) \rightarrow BOOL$
$AnalSWSTATE = \{openSW, closedSW\}$	$door\_closed \in (TRIPLE \times DOOR) \rightarrow BOOL$
$\cdots$	$\cdots$
$handle \in TRIPLE \rightarrow HSTATE$	$analogical\_switch \in TRIPLE \rightarrow AnalSWSTATE$
$gear\_extended \in (TRIPLE \times GEAR) \rightarrow BOOL$	$door\_closed \in (TRIPLE \times DOOR) \rightarrow BOOL$

The function variable $handle \in TRIPLE \rightarrow HSTATE$ captures precisely the requirement $handle_i \in \{hDown, hUp\}$ with $i \in \{1,2,3\}$. The *state* variables are the states of the gears, doors, anomalies, etc. They are modelled as follows:

$$gears\_locked\_down \in BOOL \ \wedge \ gears\_maneuvering \in BOOL \ \wedge \ anomaly \in BOOL \wedge \cdots$$

The *output* variables hold the values computed for various electro-valves:

$$general\_EV \in BOOL \ \wedge \ close\_EV \in BOOL \ \wedge \ open\_EV \in BOOL \wedge \cdots$$

The lights which indicate the position of the gears and doors to the pilot are described as *internal* variables: *greenLight*, *orangeLight*, *redLight*. These variables are bound to the output state variable *gears_locked_down* with an invariant predicate. Another *internal* variable *order* is used to record the action of the pilot on the handle.

The LG system is controlled digitally in the *normal* mode until an anomaly is detected. A permanent failure leads to an *emergency* mode where the system is controlled analogically. Accordingly the *internal* boolean variable *anomaly* is used to denote that an anomaly has been detected or not.

Step 1.2: Elicitation of the global properties of the LG system Most of the normal mode requirements are safety properties. Some identified ones are gathered in Table 1.

Step 1.3: Start with a first abstract model The first Event-B abstract model resulting from *Step 1.3*, gathers all the variables of the interface, their related invariants and initialisations. Event-B contexts are used to model the static part with the various sets and definitions that we have introduced.

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 $order$.
R_{31}	The gears outgoing event occurs if doors are open locked.
R_{41}	Opening and closing doors electro-valve are not stimulated simultaneously.
R_{51}	It is not possible to stimulate the manoeuvring EV (opening, closure, outgoing or retraction) without stimulating the general EV.

Table 1. Identified requirements

Step 1.4: The families of events of the abstract model A thorough analysis of the two action sequences (*outgoing sequence* and *retraction sequence*) described in the LG system helps us to capture the behaviour of the digital part and to derive the events. We use here state automata to make it clear the interaction between the different components (actions of the pilot, the controller, the orders received by the environment).

In the *sense event* family we have listed for example the event sense_gear to modify the input variable *gear_extended* listed above. In the same way, we have listed the other events sense_door, etc. Examples of events we have identified for the *control events* family are: stmlt_general_EV to stimulate the general electro valve, stmlt_door_opening, stmlt_gear_outgoing, stop_stmlt_general_EV, stop_ stmlt_gear_outgoing, etc. Each one modifies its related variable, for instance the event stop_stmlt_gear_outgoing sets the variable $extend\_EV$ to $FALSE$. Examples of events we classified in the *monitor events* family are: monitor_ anomaly, monitor_gears_locked_Down, monitor_ gears_manoeuvring. In the *reaction event* family we have Door_openDoor_cl2cu, Gear_extend, Gear_retract, ...

Step 2: Extension of the abstract global model with the event families

We achieve many refinement steps, by feature augmentation, to integrate gradually the variables and events related to the physical devices: the sensors, the doors and the gears.

Following *Step 2.1*, we define the behaviours of physical devices. For instance, the door behaviour 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 $doorState \in DOOR \rightarrow DSTATE$ where $DSTATE = \{ClosedLocked, ClosedUnlocked, OpenUnlocked\}$ is the enumerated set of the identified door states. The set $DOOR$ contains the three door identifiers. The function $doorState$ is a total function; this captures the requirement that all the three doors are controlled via the state transition.

The starting transition of the door behaviour is enabled by the $open\_EV$ order given by the digital part. Therefore there is a synchronisation between the digital part and the motion of the doors. We only give below the description of the starting event Door_openDoor_cl2cu; the other necessary events are similar.

event Door_openDoor_cl2cu
/* Door's Behaviour (for the three doors). The first transition of the Door Automata */
where
  @$g1$ $open\_EV = TRUE$ // all the doors Electro Valves are on
  @$g2$ $ran(doorState) = \{notOpenLocked\}$
then
  @$a1$ $doorState := DOOR \times \{notOpenNotLocked\}$ // door is being opened
end

The following event describes an event of the *control event* family.

event stmlt_gear_outgoing
/* stimulate gear outgoing electro valve once the three doors are in the open position */
where
  @$g0$ $general\_EV = TRUE$
  @$g1$ $order = hDown$
  @$g2$ $ran(handle) = \{hDown\}$
  @$g4$ $ran(door\_open) = \{TRUE\}$
  @$next$ $nextOGseq = 3$
  @$gano$ $anomaly = FALSE$ // no anomaly detected
  @$notretract$ $retract\_EV = FALSE$
then
  @$a1$ $extend\_EV := TRUE$
  @$a2$ $nextOGseq := nextOGseq + sequenceStep$
end

The variable *nextOGseq* controls the evolution of the outgoing sequence; it indicates in the event guards, the next step in the outgoing sequence. We note that the events in the *sense event* family anticipate their real future specifications, which are related to the physical part introduced later. When we have introduced the various events families and the related variables, it becomes clear for us that we have the complete control loop. Following *Step 2* the properties (listed in *Step 1.2* above) are formalised as first order predicates, integrated into the invariant of the abstract model and, proved along the horizontal refinement. As an example, the requirement R_{51} is described as follows.

$$((open\_EV = TRUE \vee close\_EV = TRUE \vee extend\_EV = TRUE \vee retract\_EV = TRUE) \Rightarrow general\_EV = TRUE)$$

To sum up, the global Event-B abstract model results from a series of refinement of contexts and machines.

Step 3: Dealing with specific properties 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. Here are some of the requirements captured in our case study: R_{22}, R_{32}, R_{42}, R_5.

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

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

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

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

In this case study, reachability is another set of the specific properties. Requirement R_1 for instance needs a specific treatment presented in the sequel. We will detail this point in Section 3.3.

3.2. Vertical Process: Building the Concrete Parts of the LG System

The vertical process includes several refinements (in *Step 4*) described below following the proposed method.

Step 4: Structural refinements of the global abstract model

In the requirement document, the inner structure of the digital part is made of two redundant computing modules. Structural refinement steps overcome the details of the behaviour of the digital part.

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

$$k\_gear\_extended \in CompModule \rightarrow ((TRIPLE \times GEAR) \rightarrow BOOL)$$

The binding between the two modules interface variables and those of the abstract module is achieved via refinements where new variables and related events are introduced.

b) Spawning the inputs inside the computing modules with refinements We introduced new events (prefixed with `spawn_`) to push the value of each *input* variable (for example *handle*) at the abstract level, in the corresponding variable (for example *k_handle*) of each computing module. As the inputs of the modules 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. The following event pattern spawns 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

We have identified a reusable **specification rule**: 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. Furthermore, the existing events, whose guards or actions involve the 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. One noticeable feature in this case is that when we have a non-deterministic event of abstract level (as for the value of the sensors), then in the refinement the event should be refined (not extended). This is another reusable **specification rule** we have identified.

c) Merging the outputs of the computing modules with refinements As depicted in Fig. 2, the k-indexed *output* variables are merged using a logical OR to set the corresponding variable 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. Several refinements are used to introduce the appropriate events.

d) Specifying the behaviour of the computing modules The two computing modules have the same behaviour which is made of: the events that monitor the system and set accordingly the state output variables and the input variables of the digital part; and the events that give orders (control decision) to the physical part through the order output variables. This results in the k-indexed form of the events related to the three categories of the interface and internal variables.

We can stop the construction of the global model at this stage; however following the guidelines provided in the method, it remains to perform the decomposition step in the basis of the *sense*, *monitor*, *control* events families (*Step 6*). Fortunately, the decomposition modules of Rodin provide assistance for this purpose. In our case where the event families structured the model, the Abrial's style of decompostion which is based on share variables [2] is the most appropriate. Indeed, the decomposition is precisely based on the families of events: the *reaction* family should be used for a (physical) machine while *sense, monitor* and *control* families should be used for another (software) machine.

As far as *Step 5* and *Step 6* are concerned, there are two main considerations: *i)* if we want to use animation capabilities on the global model, the construction should stop after the refinements of *Step 5* without doing the decomposition of the *Step 6*; *ii)* if we do not want to use animation capabilities, the construction may be continued with the decomposition process in *Step 6*. For our illustration of the case study we experiment with both considerations. First, in order to keep animation capabilities, we end our process with the *Step 5*; the *Step 6* was not performed for the case study, but only the digital part is refined with the objective to build the software part; The variables and events which are specific to the behaviour of the physical part are not refined but we keep them in the model in order to perform animation of the global model. Second, for the experimentation of the method, we go through decomposition in *Step 6*. But in this case, we have two independent models which should evolve separately, for instance the physical part may be replaced by a hardware and the control part refined into an executable code without modifying any elements of the physical part inherited from the decomposition.

3.3. Handling the Required Properties

We 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: 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.

In the following we deal with liveness (reachability) properties.

Introducing the reachability property (requirement R_{21} and R_{22}) This is a step of the horizontal refinement process (with the tag (3) in Fig. 3).
Based on the idea of Lamport's logical clocks [11], 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*; this is necessary to reason locally on relevant events. 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[1]; 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. 6). Given a set $obsEvents$ of events and a logical

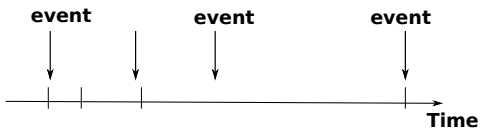

Fig. 6. Events and timestamps

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

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

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:

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

The above property expresses 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.

3.4. Experimentation with Rodin and statistics

The main modelling steps of the Landing Gear System case study have been completely achieved; that is the modelling from very abstract level to more concrete ones, the refinements and the decomposition into hardware and software parts. Applying a rigorous method as we defined, was very helpful to master the complexity of the case study.

The Rodin tool is very efficient for proving the Event-B models; a very high percentage ($\sim 87\%$) of proof obligations was automatically discharged. All the remaining proof obligations are proved interactively.

	Total	Auto	Manual	Review.	Undis.
LandingSys5	567	494	73	0	0
Abstract model					
Landing_DP_Ctx	0	0	0	0	0
LandingSysDP_A	109	106	3	0	0
LandingSysDP_SWITCH_A	3	3	0	0	0
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	42	34	8	0	0
LandingSysDP_DGT_R2_INOUT	56	40	16	0	0
LandingSysDP_DGT_R3_DG	234	188	46	0	0

Table 2. Statistics of PO generated and proved with Rodin

The specifications are available online[2]. The current version of the Event-B models is deliberately partial as we chose to focus on representative events instead of being exhaustive. We have used the version 3.4 of Rodin in the last experimentations; the statistics on Proof Obligations are given in Table 2.

The proofs discharged using the interactive prover are related to the structural refinement and specifically they are related to the binding invariants.

[2] `hencher.ls2n.fr`

Using the Rodin tool we have modelled and refined the Landing Gear System until to take account of the main requirements about software part, physical part and some of the specific properties as explained in Section 3.3. After several steps of vertical refinements we have a complete model of the Landing Gear system. For the experimentation purpose, using the Event-B decomposition technique [22], we decompose the last model of the system into two parts corresponding to the hardware part and the software or control part. The so-called A-style decomposition, based on the separation of events through different machines, and implemented as a Rodin plugin [15], was successfully used in this step.

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 can be of finer grain.

Moreover a good naming discipline is necessary at each level of the modelling. It helps for traceability and to face the complexity due to the size of the model.

As far as the ProB animation tool (integrated in Rodin) is concerned, it is very helpful to tune the Event-B models; indeed the failure in the animation gives information about the (bad) states and accordingly the wring part of the model can be rewritten.

3.5. Tooling Concerns

During the above experimentations, we felt need assistance at different steps for different motivations. We mention some situations where tooling would be helpful, in addition to Rodin's facilities, to apply the Heñcher method.

- RT_1 *Starting the process.* The first steps are often crucial when applying a method. Assistance is required to answer the users's question *How to start the process?*.
- RT_2 *Incremental step-by-step refinement.* The Heñcher method is tightly based on refinements. To master the development process in Event-B, the recommended approach is to proceed with small and well-defined refinement steps; that means the complexity is not in individual refinements but in the whole refinement process; but there is a lack of assistance tool to help developers. For instance the developer may be happy with a highlighting of some parts of its specifications.
- RT_3 *Pattern-based substitutions and automatic refinements* for composition and physical part refinement. This requirement needs a full development when the specifications are in Rodin.
- RT_4 *Team collaboration.* Making it easy for several people to work simultaneously on a project, versioning, decision traceability are all important concerns when dealing with big Event-B projects as the one we have studied.
- RT_5 *Overall development process management.* When one should stop with a development step? Is the current state sufficient to start the next step? there is a need of various metrics to evaluate the quality of ongoing specifications (completeness,...).
- RT_6 *Iterative process of model evolution.* Often, one wants to modify an abstract model (for example adding a variable or an event in the M0 machine), and has the modification be systematically propagated in the chain of the remaining models and refinements. The tool supporting the method should enable such a continuous model evolution.

RT_7 *Traceability of the refinement chain of a single event.* During the development and proof steps, a practical concern is to review the chain of refinement of a single event without all the surrounding events. An assistance tool is also needed.

RT_8 *Securing copy-paste operations.* It is often the case, to copy-paste similar events. This is particularly true when we have systems with redundancies of several instance of the same objects, like the gears, the doors and the sensors in our case study. Due to unavoidable human errors, lot of time is spent fighting again undischarged proof obligations. There is a need of a parameterized refactory tool that for instance rewrites an existing event by substituting some variables with others.

We undertake the development of an assistance tool to provide the main functionalities we have identified during experimentations. In the next section we introduce the basis of the design of a web companion tool we have developed to address the requirements RT_1 and RT_2.

4. An Assistance Tool

In this section we introduce our proposed prototype tool to assist the users of the Heñcher method. The tool is designed as a companion tool of the existing frameworks such as Rodin or Atelier B. It helps the users to apply the Heñcher method, and to build more quickly the preliminary Event-B models, which will be analysed in the dedicated existing environments.

From the provided interface of a given control system the tool generates in an incremental way, following the steps of the Heñcher method, the Event-B abstract models. The tool is designed for Event-B users (specifiers/engineers). The inputs of the tool will be provided by the users; an user-friendly graphical interface is designed for this purpose. The tool provides the development guidelines of the Heñcher method and some skeletons of Event-B models as output.

Generation of an Event-B models from a control system interface On the basis of the interface between a control system and its environment as presented in Section 2, a specifier should provide the interface variables, the list of controlled devices of its system, the global and specific properties required by its systems. From these elements the specifier will be assisted in building an Event-B machine M_0 and then a refined one M_1. The machine M_1 can be further refined until more concrete levels but one has to use the dedicated tool (Rodin for instance).

The collected interface Let an interface made of $\mathcal{X}_s$ a set of the input variables, $\mathcal{X}_o$ a set of the output variables, $\mathcal{X}_c$ a set of the control variables. In addition, let $\mathcal{X}_i$ be a set of internal variables of the controller; (a part of the control variables are used for the feedback, the internal variables are the part of the control variables used by the controller but which are not output as feedback). According to the system at hand, the user should define the types of each of the previous variables. That means each variable of $\mathcal{X}_s$ has its type in $\mathcal{T}_s$. Therefore for each family X of variables of the interface and the related set of types T_X, we have a type mapping $\{(\sigma_i, \tau_i)\} \subseteq X \times T_X$.

Assume then three type mappings built by the user from its system requirements and from the variables $\mathcal{X}_s$, $\mathcal{X}_o$, $\mathcal{X}_c$ and the related sets of types T_{X_s}, T_{X_o}, T_{X_c}.

4.1. An Overview and the Design of the Tool Assistant

The flowchart in Fig. 7 gives an overview of how the tool assists the user in building the abstract model in a global process. We will then define the steps of this global process.

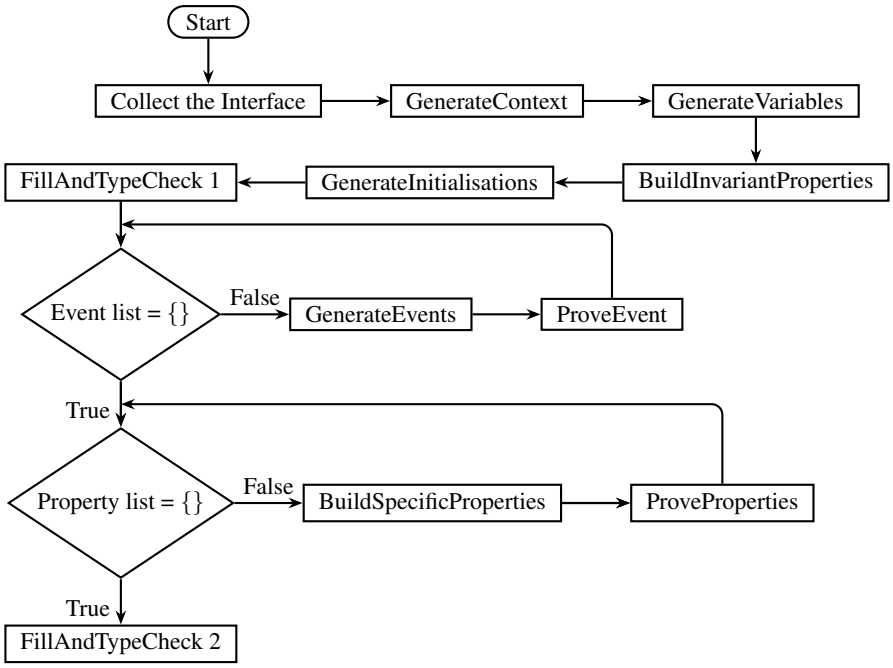

Fig. 7. Flowchart of the global functioning of the tool

Stepwise building of the Event-B model

Building the context of the model. From the extracted sets of types T_{X_s}, T_{X_o}, T_{X_c}, a context Ctx0 made of the sets coming from T_{X_s}, T_{X_o}, T_{X_c} is built. Each carrier set of Ctx0 comes from $T_{X_s} \cup T_{X_o} \cup T_{X_c}$ (see Fig. 8).

context $Ctx0$
constants
    to be completed by the specifier
sets
    Each element of the union $T_{X_s} \cup T_{X_o} \cup T_{X_c}$
axioms
    to be completed by the specifier
end

Fig. 8. Event-B context skeleton

The initial Event-B abstract model (Step 1 of the method). The skeleton of the Event-B abstract model (M_0) to build is depicted in Fig. 9.

The functions GenerateVariables, BuildInvariantProperties, BuildSpecificProperties and GenerateInitialisationSubstitutions are used to compute respectively the set of variables, typing invariants, specific invariants and default initialisations for the M_O model. They are elementary functions; most of them traverse a set, and encode in Event-B syntax the elements of the sets.

Machine M_0
SEES Ctx0
VARIABLES
GenerateVariables$\mathcal{X}_s \cup \mathcal{X}_o \cup \mathcal{X}_c \cup \mathcal{X}_i$
INVARIANTS
BuildInvariantProperties$(\mathcal{X}_s, \mathcal{X}_o, \mathcal{X}_c, \mathcal{X}_i)$
BuildSpecificProperties$(\mathcal{X}_s, \mathcal{X}_o, \mathcal{X}_c, \mathcal{X}_i, \mathcal{P}_s, \mathcal{P}_n, \mathcal{P}_l)$
INITIALISATION
GenerateInitialisations$(\mathcal{X}_s, \mathcal{X}_o, \mathcal{X}_c, \mathcal{X}_i)$
END

Fig. 9. Event-B model skeleton

The function GenerateInitialisations$(\mathcal{X}_s, \mathcal{X}_o, \mathcal{X}_c, \mathcal{X}_i)$ works as follows: for each variable v in $\mathcal{X}_s \cup \mathcal{X}_o \cup \mathcal{X}_c \cup \mathcal{X}_i$, if T_v is a set in Ctx0 corresponding to the type of v then a substitution `v :: T_v` is generated. Note that these default initialisations can be modified by the users to meet its needs.

The global properties of the system. Considering the requirements of the given system let $\mathcal{P}_s$ be the set of the safety properties, $\mathcal{P}_n$ be the set of non-functional properties, and $\mathcal{P}_l$ be the set of liveness properties. Each property should be formalised by the user and incorporated with the assistance tool in the model under construction.

Assistance in horizontal refinement steps

Construction of the events of the abstract model (Step 2 of the method). The current Event-B abstract model is now extended (that means feature augmentation) with the previous family of events (sensing events, monitoring events, control events).

Let $\mathcal{E}_s$ be the set of sensing events, $\mathcal{E}_m$ be the set of monitoring events, $\mathcal{E}_c$ be the set of control events, $\mathcal{E}_a$ be a set of reaction events.

The algorithm of GenerateEvents$(\mathcal{E}_s, \mathcal{E}_m, \mathcal{E}_c)$ is listed in Fig. 11s:

Machine M_0
...
EVENTS
GenerateEvents$(\mathcal{E}_s, \mathcal{E}_m, \mathcal{E}_c)$
END

Fig. 10. Event-B model M_0

while $(\mathcal{E}_s \neq \varnothing) \wedge (\mathcal{E}_m \neq \varnothing) \wedge (\mathcal{E}_c \neq \varnothing)$ **do**
Select an event e from $\mathcal{E}_s$ and update $\mathcal{E}_s$ $(E_s = E_s - \{e\})$
or
Select an event e from $\mathcal{E}_m$ and update $\mathcal{E}_m$
Select an event e from $\mathcal{E}_c$ and update $\mathcal{E}_c$
$M_0 = AddEvent(M_0, BuildEvent(e))$
end while

Fig. 11. Events construction adding algorithm

The function $BuildEvent(e)$ generates a skeleton for each event name e. This skeleton should be filled by the user.

The Event-B abstract model resulting from this stage should now be extended with the specific properties identified by the user in the system requirements (they have to be formalised by the user).

We build a web application to implement the starting sequence of the process depicted in Fig. 7. Its application to a part of the case study is shown in Fig. 12 for the description of variables. Other informations on the tool can be found on the dedicated website[3].

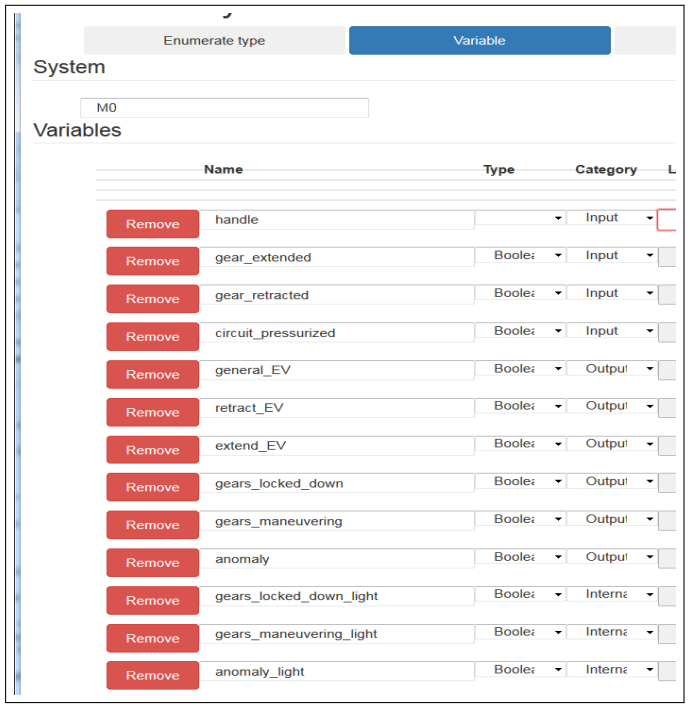

Fig. 12. Interface of the web assistant to help in describing the abstract state

Adding properties to the model is a refinement process (Step 3 of the method). The informal properties named and described by the user have to be formalised and integrated in the abstract model. The assistance here consists in selecting for formalisation, each property from those listed in $\mathcal{P}_s$by the user (see Fig. 13).

while $(\mathcal{P}_s \neq \varnothing)$ **do**
  Select a property p from $\mathcal{P}_s$
  $\mathcal{P}_s = P_s - \{p\})$ // update $\mathcal{P}_s$
  $M_0 = AddProperty(M_0, Formalise(p))$
  Prove M_0
end while

Fig. 13. Properties adding algorithm

The function $Formalise(p)$ enables one to edit (through a popup for instance) each involved property. In the same way, the other specific (reachability and non functional) properties listed in $\mathcal{P}_n$ are integrated into the refined model M_1 using the algorithm described in Fig. 14. Assistance is provided to the user for selecting the specific properties to be integrated into the abstract model.

[3] hencher.ls2n.fr

$M_1 = M_0$ // initially M0 is copied; then updated.
while $(\mathcal{P}_n \neq \varnothing)$ **do**
  Select a property p from $\mathcal{P}_n$ and update $\mathcal{P}_n$
  $M_1 = Add_p(M_1, formalise(p))$
end while

The function *Formalise(p)* is as previously defined. $\mathcal{P}_n$ is the set of non-functional properties.

Fig. 14. Additional non-functional properties

Fig. 15 illustrates three of the requirements as listed in the web interface of the assistance tool; they appear in the Rodin snapshot depicted in Fig. 16.

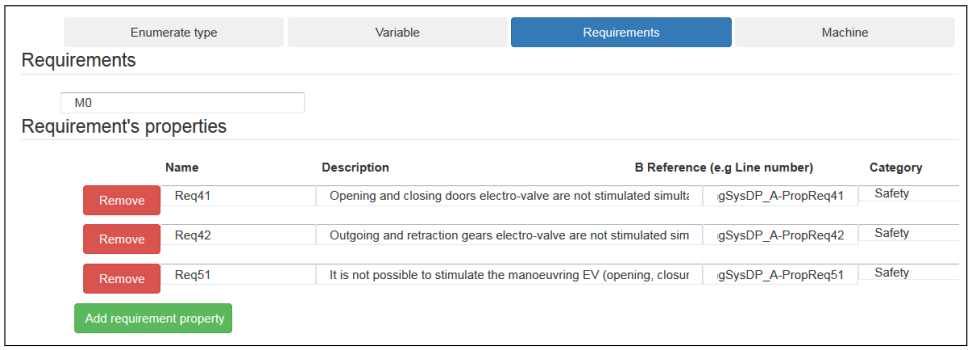

Fig. 15. Informal required properties listed in the assistance tool

To improve traceability between, it is better to edit directly the property in Rodin and to use its label to tag the property in the list of the informal properties.

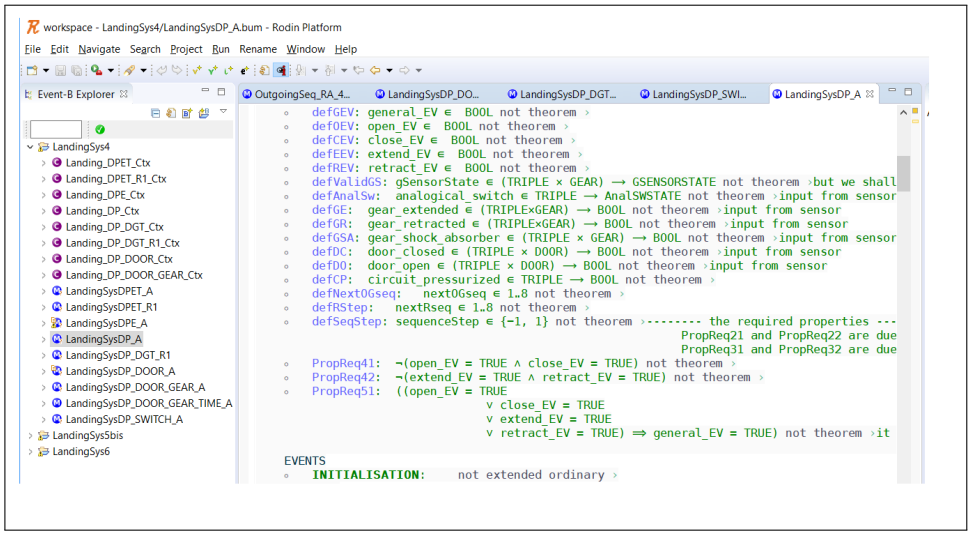

Fig. 16. Required properties formalised in Rodin interface

The remaining steps of the method, that is the refinement of the control software (**Step 6.1** of the method) and the refinement of the controlled environment (**Step 6.2**) of the

methods should be achieved within Rodin (or a dedicated Event-B framework). However we propose some assistance as described in the following section.

Assistance in the vertical refinement process (*Step 4, 5*) There is no specific tool for the *Step 4. Refining the global abstract model*. This steps consists in describing the behaviour of the devices involved in the case study. This can be done by using automata or the appropriate models; by encoding these models in Event-B (roughly, the transitions of the automata are encoded as Event-B events).

Decomposition into software and physical parts (Step 5). The methodological guide to achieve the decomposition is as follows: the digital part is made with all the events defined in the *sense events* ($\mathcal{E}_s$), the *monitor events* ($\mathcal{E}_m$) and the *stimulate events* ($\mathcal{E}_c$) families whereas the physical environment gathers all the events defined in the *reaction events* ($\mathcal{E}_a$) families.

Accordingly we systematically provide the user with the lists of the events that she/he must select for the decomposition process. The events of the control part are computed as: $\mathcal{E}_{soft} = \mathcal{E}_s \cup \mathcal{E}_m \cup \mathcal{E}_c$; those of the physical part are computed as: $\mathcal{E}_{phys} = \mathcal{E}_a$. These two lists are then used as the input of the decomposition plugin of Rodin.

The decomposition in Event-B consists in selecting and separating the desired events into two machines. In our case the model $M_{control}$ corresponding to the control part is the projection of M_1 on $\mathcal{E}_{soft}$. Similarly the model M_{phys} corresponding to the physical part is the projection of M_1 on the list $\mathcal{E}_{phys}$ (See Fig. 17).

The Rodin tool already provides a decomposition plugin [15] that performs the projection of the provided machine according to the selected events. Therefore our assistance tool provides to its users, the lists of events that he/she should select when using the decomposition plugin of Rodin.

Fig. 17. Model decomposition skeleton

At this stage our tool provides much assistance to begin with the modelling in Event-B using Rodin; but many other tool facilities are provided with the Rodin to help its users. In the following we show how some of these tools can be used at various stages of our proposed method to satisfy the tool requirements identified in Sect. 3.5.

4.2. Tool Requirements Handled by Existing Rodin Plugins

We have experimented with some Rodin plugins to put our method in practice. But there are many other plugins available for Rodin to extend and complete the features of the

Rodin. We have studied them, and in the following we report on how some of these existing plugins can help one to apply the Heñcher method by resolving some of the previously mentioned requirements (see Section 3.5). Therefore for each of the tool requirements we advice the available or candidate plugins.

RT_2 Incremental step-by-step refinement. Our tool already provides a preliminary assistance in this direction by considering the families of events to be used in each refinement step.But more remain to be done to gain more flexibility within Event-B; there is a prototype plugin on *Group refinement* [4] which targets the simplification of the refinement links between abstract machines and their refinements; the idea is to relax the constraints on introducing dummy variables and their related housekeeping events that are there only to satisfy the refinement relation.

RT_3 Pattern based substitutions and automatic refinements. Some plugins propose to add (de-)composition features into Event-B/Rodin. The *Feature composition* plugin [17] enables one to merge Event-B machines and their seen contexts with the facilities to highlight multiple declarations of variables or events and to resolve conflicting elements.

RT_4 Collaboration: working in parallel, versioning, decision traceability. There exists a plugin called "team-based development" [19] which allows Event-B models to be stored in a SVN repository. That plugin allows to share the Event-B specifications but not the proof effort. The *Modularisation* plugin [5] may also help here by allowing separate development of part of the a global system as sub-modules and then by combining them. Indeed, this plugin enables one to weave together modules composing a model so that they work on the same global problem.

R_5 Overall development process management. An experimental plugin named *Model critic* [14] could be used to evaluate models using informal heuristics of what is typically a bad practice in model construction. That is a first step to evaluate quality of Event-B specifications, but it is not enough, to use it into an industrial process.

RT_6 Iterative process of model evolution. We identified several plugins that can help in fulfilling this requirement. The *Refactory* plugin [18] provides functionalities to rename the declaration and all the occurrences of an element of an Event-B model without modifying its proof state. There is a need for applying the same principle for more complex operations; for instance the insertion of event and its basic refinement through the chain of refinements, the detection and replacement of all the occurrences of an event in a model and trough its refinements, but with the smallest impact on the proof effort. The *Transformation patterns* plugin [20] could be used to automatise the different steps of the Heñcher method. From a given Event-B machine, we can produce a new one by applying some "transformations" : adding new variable, new event, new invariant, finding and using some event's guards, ask the user to enter some missing information, etc. No information is given about the necessary proof effort when using this plugin. The *Design pattern* plugin [16] is dedicated to reuse former development (as a pattern) in a new development by

[4] `http://wiki.event-b.org/index.php/Group_refinement_plugin`

[5] `wiki.event-b.org/index.php/Modularisation_Plug-in`

matching the corresponding variables and events. This results in a refined machine which embeds the former development. The proofs of the used pattern would be reused too. The correctness of the matching is only syntactically checked.

RT_7 Traceability of the refinement chain of an event. We have not yet identified any tool or plugin to help for this concern.

RT_8 Securing copy-paste operations. We have not yet identified any tool or plugin to help for this concern.

Issues on tool development and maintenance We are aware of the effort to be done to face the recurrent issues on tool development and maintenance. Most of the mentioned plugins are experimental, and insufficiently documented; they are not all based on the same (up-to-date) versions of Rodin. Many of them are difficult to install because of the (incompatible) required dependencies.

Accordingly, the *open source* policy is a solution to share the development and maintenance effort. We envision this solution for the tools we are developing and for the effort to be devoted to the remaining identified requirements.

5. Conclusion

In this paper we focused on the application of the Heñcher method to the Landing Gear Case study and paid much attention to the necessary tool support for the method. This led to the detailed presentation of the design of a companion tool of our method. Then we showed how the tool was used together with the Rodin tool and plugins to experiment with the case study. This work extends substantially our previous work presented in [4] where the proposed method Heñcher was introduced. We proposed the method Heñcher to guide step by step the construction of embedded control systems with Event-B. We built on the well-known structure of control systems and on the experiments of several case studies where the Event-B was used and where some methodological guidelines was provided [7,6,21]. We provided preliminary assessment in [4]; in this paper the case study is dealt with more details and we covered all the steps of the proposed method. The Landing Gear case study is representative of large systems involving the control and the interaction between software and physical parts. Before going into the presentation of the assistance tool that we have developed, we emphasised the motivations and the requirements for the needed tools.

Among the identified tool requirements, we detailed a tool to assist the specifiers in starting the modelling process, and to assist them during the development process with Rodin. Yet our Heñcher assistance tool is a prototype developed as a standalone web application. We experimented with Rodin but, it can be used not only for Rodin.

We have studied other available Rodin plugins that can help in putting into practice or to implement the remaining identified tool requirements.

Apart from the development of the tools related to facilitating the reuse of existing plugins (template reuse, refactoring, injection of events, etc), the short-term perspectives of our work are to continue the development of the Heñcher tools and their experimentation

with other large scale case studies in combination with the Rodin platform. Our experiment with the Landing Gear case study which has the main features of cyber-physical systems, will be exploited to deal with more case studies of this category. Indeed there are other challenges to tackle in this area, such as handling real-time properties, dealing with the construction of systems built on the basis of digital twins concepts.

The long-term perspective is the provision of a generic development pattern related to embedded system which will make it more easier to describe requirements and properties and get the major part of the development generated.

Acknowledgments. Thanks to Léo Cassiau, Geoffrey Desbrosses, Alexis Giraudet, Jean-Christophe Guérin, Asma Khelifi, Ugo Mahey and Tantely Randriamaharavomanana, Master students at University of Nantes in 2017, who worked with us on the tools.

References

1. Abrial, J.R.: Modeling in Event-B - System and Software Engineering. Cambridge University Press (2010)
2. Abrial, J.R., Hallerstede, S.: Refinement, Decomposition, and Instantiation of Discrete Models: Application to Event-B. Fundam. Inform. 77(1-2), 1–28 (2007)
3. Alkhammash, E., Butler, M.J., Fathabadi, A.S., Cîrstea, C.: Building Traceable Event-B Models from Requirements. Sci. Comput. Program. 111, 318–338 (2015), https://doi.org/10.1016/j.scico.2015.06.002
4. André, P., Attiogbé, C., Lanoix, A.: Systematic Construction of Critical Embedded Systems Using Event-B. In: Abdelwahed, E.H., Bellatreche, L., Benslimane, D., Golfarelli, M., Jean, S., Méry, D., Nakamatsu, K., Ordonez, C. (eds.) New Trends in Model and Data Engineering - MEDI 2018 International Workshops, DETECT, MEDI4SG, IWCFS, REMEDY, Marrakesh, Morocco, October 24-26, 2018, Proceedings. Communications in Computer and Information Science, vol. 929, pp. 200–216. Springer (2018), https://doi.org/10.1007/978-3-030-02852-7_18
5. Boniol, F., Wiels, V.: The landing gear system case study. In: Boniol, F., Wiels, V., Ait Ameur, Y., Schewe, K.D. (eds.) ABZ2014. CCIS, vol. 433, pp. 1–18. Springer International Publishing (2014)
6. Damchoom, K., Butler, M.J.: Applying Event and Machine Decomposition to a Flash-Based Filestore in Event-B. In: 12th Brazilian Symposium on Formal Methods, SBMF 2009. LNCS, vol. 5902, pp. 134–152. Springer (2009)
7. Damchoom, K., Butler, M.J., Abrial, J.R.: Modelling and Proof of a Tree-Structured File System in Event-B and Rodin. In: 10th International Conference on Formal Engineering Methods, ICFEM 2008. LNCS, vol. 5256, pp. 25–44. Springer (2008)
8. Hoang, T.S., Snook, C.F., Fathabadi, A.S., Butler, M.J., Ladenberger, L.: Validating and verifying the requirements and design of a haemodialysismachine using the rodin toolset. Sci. Comput. Program. 158, 122–147 (2018), https://doi.org/10.1016/j.scico.2017.11.002
9. Jard, C., Roux, O.H. (eds.): Communicating Embedded Systems: Software and Design. Wiley-ISTE (2009)
10. Karsai, G., Sztipanovits, J., Ledeczi, A., Bapty, T.: Model-integrated development of embedded software. Proceedings of the IEEE 91(1), 145–164 (Jan 2003)
11. Lamport, L.: Time, Clocks, and the Ordering of Events in a Distributed System. Commun. ACM 21(7), 558–565 (1978)
12. Méry, D., Singh, N.K.: Formal Specification of Medical Systems by Proof-Based Refinement. ACM Trans. Embedded Comput. Syst. 12(1), 15 (2013)

13. Parnas, D.L., Madey, J.: Functional Documents for Computer Systems. Science of Computer Programming 25(1), 41–61 (1995), citeseer.ist.psu.edu/parnas95functional.html
14. Event-b rodin platform plug-ins: Model critic plug-in, http://wiki.event-b.org/index.php/Model_Critic, accessed: 2019-03-14
15. Event-b rodin platform plug-ins: Decomposition plug-in, http://wiki.event-b.org/index.php/Decomposition_Plug-in_User_Guide, accessed: 2019-03-14
16. Event-b rodin platform plug-ins: Design pattern, http://wiki.event-b.org/index.php/Pattern, accessed: 2019-03-18
17. Event-b rodin platform plug-ins: Feature composition plug-in, http://wiki.event-b.org/index.php/Feature_Composition_Plug-in, accessed: 2019-03-14
18. Event-b rodin platform plug-ins: Refactoring framework, http://wiki.event-b.org/index.php/Refactoring_Framework, accessed: 2019-03-14
19. Event-b rodin platform plug-ins: Team-based development, http://wiki.event-b.org/index.php/Team-based_development, accessed: 2019-03-11
20. Event-b rodin platform plug-ins: Transformation patterns, http://wiki.event-b.org/index.php/Transformation_patterns, accessed: 2019-03-18
21. Satpathy, M., Ramesh, S., Snook, C.F., Singh, N.K., Butler, M.J.: A Pixed Approach to Rigorous Development of Control Designs. In: 2013 IEEE International Symposium on Computer-Aided Control System Design, CACSD 2013, Hyderabad, India, August 28-30, 2013. pp. 7–12. IEEE (2013), https://doi.org/10.1109/CACSD.2013.6663474
22. Silva, R., Butler, M.: Shared Event Composition/Decomposition in Event-B. In: 9th International Symposium onFormal Methods for Components and Objects, FMCO 2010. LNCS, vol. 6957, pp. 122–141. Springer (2012)
23. Singh, N.K., Wang, H., Lawford, M., Maibaum, T.S.E., Wassyng, A.: Stepwise formal modelling and reasoning of insulin infusion pump requirements. In: Duffy, V.G. (ed.) Digital Human Modeling - Applications in Health, Safety, Ergonomics and Risk Management: Ergonomics and Health - 6th International Conference, DHM 2015, Held as Part of HCI International 2015, Los Angeles, CA, USA, August 2-7, 2015, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9185, pp. 387–398. Springer (2015), https://doi.org/10.1007/978-3-319-21070-4_39

Pascal ANDRE received his Ph.D. from the University of Rennes I in 1995. He joined the Faculty of Sciences of Nantes in 1996 and then, spent 4 years as assistant professor at INP-HB engineering school (Ivory Coast). He is currently Associate Professor at the University of Nantes, France. He carries out his research activities in the Reliable Architecture and Software (AeLoS) team, at the laboratory of digital sciences of Nantes (LS2N). He published a series of course books on information system design in french. His main research topics concern the use of formal methods and verification techniques for software modelling and analysis, particularly in the context of component-based systems. He also actively work on rigorous approaches for model-driven engineering and reverse engineering.

Christian ATTIOGBÉ received the Ph.D. degree in Computer Science from the University of Toulouse, Toulouse, France, in 1992. He joined the Faculty of Sciences of Nantes in 1994 and he is currently Professor at the University of Nantes, Nantes, France. His research interests include formal approaches for software modelling and analysis, correct-by-construction using refinement, embedded-systems and heterogeneous systems modelling. He published several peer-reviewed papers on these topics. He is the leader of the

Reliable Architecture and Software (AeLoS) team, at the laboratory of digital sciences of Nantes (LS2N) since 2007.

Arnaud Lanoix received his Ph.D. from the University of Franche-Comté in 2005. He spent 3 years as a post-doctorate at the LORIA lab (Nancy, University of Lorraine). He is Associate Professor since 2008 at the Université de Nantes and carries out his research activities in the Reliable Architecture and Software (AeLoS) team, at the laboratory of digital sciences of Nantes (LS2N). His main research topics concern the use of formal methods and verification techniques for software modelling and analysis, particularly in the context of component-based systems.

Received: May 1, 2019; Accepted: September 13, 2019.