Modeling and verifying an arrival manager using the formal Event-B method

Amel Mammar, Michael Leuschel

Software Tools for Technology Transfer manuscript No.
(will be inserted by the editor)

Modeling and Verifying an Arrival Manager using Event-B ⋆

Amel Mammar[1], **Michael Leuschel**[2]

[1]SAMOVAR, Télécom SudParis, Institut Polytechnique de Paris, France
[2]Institut für Informatik, Universität Düsseldorf

Received: date / Revised version: date

Abstract. The present paper describes an Event-B model of the Arrival MANager system (called AMAN), the case study provided by the ABZ'23 conference. The goal of this safety critical interactive system is to schedule the arrival times of aircraft at airports. This system includes two parts: an autonomous part which predicts the arrival time of an aircraft from external sources (flight plan information, radar and weather information, etc.) and an interface part that permits the Air Traffic Controller (ATCo) to submit requests to AMAN like changes regarding the arrival times of aircraft. To formally model and verify this critical system, we use a correct-by-construction approach with the Event-B formal method and its refinement process. We mainly consider functional features of the case study; all proof obligations have been discharged using the provers of the Rodin platform under which we carried out our development. To help users understand how AMAN works and its main functionalities, a visualisation of the Event-B models was achieved using the VisB component of ProB. Our models have been validated using ProB by checking scenarios related to different functional aspects of the system.

Key words: System modeling, Event-B method, Refinement, Verification

1 Introduction

In this paper, we introduce a formal model of the Arrival MANager system (called AMAN). This system has been provided as a case study in the context of the ABZ'23 conference. The main objective of the AMAN system is to help the Air Traffic Controller (ATCo) manage the arrival of aircraft approaching the considered airport by providing it with an arrival sequence. To predict the arrival times of aircraft, AMAN uses external sources like flight plan data, radar data, weather information, etc. The process of calculating concrete arrival times itself is out of the scope of this paper, only its output is considered.

The AMAN system works in collaboration with the ATCo who can suggest some modifications on the arrival sequence to the AMAN. The ATCo can also block periods of time (for runway cleaning for instance) notifying the AMAN that these time slots are no longer available; any predicted arrival corresponding to these slots must be thus moved. The ATCo also has the possibility to put an already predicted aircraft on hold, informing the AMAN that this latter must be removed from the arrival sequence. Finally, the interface permits the ATCo to focus on specific aircraft that are predicted to land within the next minutes (between 15 and 45) by selecting a zoom level. In that case, only these related aircraft are displayed within the interface.

In [22], a first formal model of the AMAN system using the Event-B formal method has been presented. The current paper describes a new Event-B model of the same system with fewer variables and more natural expression of the requirements. The use of the Event-B formal method with its refinement technique permits to master the complexity of a system by gradually introducing its different elements and characteristics. Building a formal model of such a system permits to verify the expected properties including the safety ones. Our approach to model this system follows the four-variable model of Parnas and Madey [26] that distinguishes two classes of variables, environment and controller variables (see Section 2). The first class denotes

Send offprint requests to: Amel Mammar, E-mail: amel.mammar@telecom-SudParis.eu

⋆ This work was supported in part by the ANR projet DISCONT.

the elements that are outside the controller and whose states are read by the sensors. The state of the environment variables are updated when the system sends commands to them through the actuators. The second class includes the inputs and the outputs of the system. To master the complexity of this system, we deal with one environment/controller variable at each refinement step.

The Event-B models have been built by the first author. Her past industrial experience in the formal specification and verification of railway interlocking systems, in collaboration with Thales and RATP, helped her in this task. The knowledge acquired during this experience has been put in practice and reinforced through the different modelling of the previous ABZ case studies [24,20,21,19]. The development of the Event-B models took two months and has been done using the Rodin platform [12] that provides editors, provers and several other plugins for various tasks like animation and model checking with ProB [17]. Our Event-B model considers most of the requirements except those related to the interface appearance (See Table 1). In this paper, we used ProB in order to animate the built models with three purposes: discovering the potential scenarios (a sequence of events) that violate the invariant before going in a hard/long proof phase, validating the specification by simulating the provided scenarios in order to be sure that we have specified the right system, and constructing a visualisation of the Event-B models using the VisB component of ProB.

1.1 Event-B method

Event-B [3] is the successor of the B method [2] for developing correct discrete systems using mathematical notations. This method is well-suitable for complex systems thanks to its refinement concept that permits incrementally introducing details by starting with an abstract view of model that it is then enriched until obtaining a model that includes all the elements of the system.

An Event-B model includes two elements: *context* and *machine*. A context describes the static part of an Event-B specification; it consists of constants C and sets S (user-defined types) together with axioms A that specify their properties. The dynamic part of a system is included in a machine that defines variables V and a set of events E. The possible values that the variables hold are restricted using an invariant, denoted Inv, written using a first-order predicate on the state variables. Each event, of the form (**ANY** X **WHEN** G **THEN** Act **END**), is enabled when all the conditions G, named guards, prior to its triggering hold. When several events are enabled, only one is triggered. In this case, substitutions Act, called actions, are applied over variables. In this paper, we restrict ourselves to the *becomes equal* substitution, denoted by $(x := e)$.

Refinement is a process of enriching or modifying a model in order to augment the functionality being modeled, or/and explain how some purposes are achieved. Both Event-B elements *context* and *machine* can be refined. A context can be extended by defining new sets S_r and/or constants C_r together with new axioms A_r. A machine is refined by introducing new variables and/or replacing existing variables with new ones V_r that are typed with an additional invariant Inv_r. New events can also be introduced to implicitly refine a **skip** event. In this paper, the refined events have the same form **ANY** X_r **WHEN** G_r **THEN** Act_r **END**. The semantics of the Event-B notations used in this paper is provided in Appendix B.

To ensure the correctness of the specification, proof obligations are generated for the first abstract model and for each refinement level. These proof obligations aim at proving invariant preservation by each event, but also to ensure that the guard of each refined event is stronger than that of the abstract event. These guard strengthening refinement proof obligations ensure that event parameters are properly refined. More information on proof obligations can be found in [3]. Basically:

1. For each event, we have to establish that its triggering maintains the invariant, that is :

$$\forall\ S,\ C,\ X.\ (A \wedge G \wedge Inv \wedge BA_{Act} \Rightarrow Inv')$$

   where BA_{Act} is the before-after predicate of the actions of an event and Inv' is the invariant applied to the after values.
2. To prove that a refinement is correct, we have to establish the following two proof obligations:
   - *guard strengthening*: the guard of the refined event should be stronger than the guard of the abstract one:
     $\forall(S, C, S_r, C_r, V, V_r, X, X_r).$
     $(A \wedge A_r \wedge Inv \wedge Inv_r \Rightarrow (G_r \Rightarrow G))$
   - *Simulation*: the effect of the refined action should be stronger than the effect of the abstract one:
     $\forall\ (S,\ C,\ S_r,\ C_r,\ V,\ V_r,\ X,\ X_r).$
     $(A \wedge A_r \wedge Inv \wedge Inv_r \wedge G_r \wedge BA_{Act_r} \Rightarrow$
     $BA_{Act} \wedge {Inv_r}')$
   - *Convergence*: convergence of new events is optional and has not been used in this case study because it was not necessary, new events being allowed to loop forever, according to the requirements.

To discharge the proof obligations, we used the automatic provers integrated to the Rodin platform[1] and also the external SMT [14] and Atelier B [11] provers that we have imported as plugins to Rodin. The Rodin platform also offers a theory component that allows us to extend the mathematical concepts of Event-B by defining user-types and their associated operators [7]. The use

[1] `http://www.event-b.org/install.html`

of the theory component also permits to cope with the complexity of a system by importing and reusing existing theories.

1.2 The ProB model checker

In this paper, we used the animator and model checker ProB [18] for validating our models in order to ensure that they are error-free and their behaviors correspond to the expected ones.

ProB[2] has been developed over twenty years and contains a model checker for verifying LTL (Linear Temporal Logic) [27] and CTL (Computational Tree Logic) [10] properties. The core of ProB is written in Prolog; its purpose is to be a comprehensive tool in the area of formal verification methods. Its main functionalities can be summarized as follows.

1. ProB permits to exhibit scenarios (a sequence of events) that, starting from a valid initial state of the system, reach a state that violates its invariant.
2. ProB allows the animation of the B/Event-B models, that is, the user can simulate different scenarios from a given starting state that satisfies the invariant. Via its graphical user interface, the animator provides the user with: (*i*) the current state, (*ii*) the history of the event triggering that has led to the current state and (*iii*) a list of all the enabled events, along with proper parameters instantiations. In this way, the user does not have to guess the right values for the event parameters.
3. ProB allows to detect absolute/relative deadlocks,
4. ProB provides many features to visualise an individual state or the entire state space.

1.3 Contributions

The contributions of this paper with respect to [22] are as follows:

- a more detailed description of the modeling strategy used (see Section 2);
- A new Event-B model which defines fewer variables and more natural expression of the different invariants. In this model, we timestamp each variable with the current time (see Section 3);
- additional examples and explanations about the verification phase using model-checking and scenarios (see Section 4.3);
- more details about the development of the visualisation (see Section 4.4);
- a comparison with similar solutions using different approaches/formal languages (see Section 5).

[2] `https://prob.hhu.de/`

1.4 The structure of the paper

The rest of this paper is structured as follows. Section 2 presents our modelling strategy. Then, Section 3 describes our model in more details. The validation and verification of our model are discussed in Section 4 along with a visualisation of the model using the VisB component of ProB. Section 5 compares our specification with similar specifications that use different languages/approaches. Finally, Section 6 concludes the paper.

2 Requirements and modeling strategy

This section presents the principles of control systems and an overview of the Event-B architecture corresponding to the modeling of the AMAN system.

2.1 Control abstraction

In this paper, we use the concepts described by Parnas and Madey in [26]. The AMAN system can be considered as a control system that reads information from the environment elements (e.g., the desired arrival time of an aircraft, radar information) using sensors and uses a set of actuators to transmit the adequate orders to these elements (see Figure 1). A sensor measures the value of an environment element m, called a *monitored* variable (*e.g.*, the desired arrival time of an aircraft, radar information), and provides this measure (*e.g.*, the desired hour/minute) to the software controller as an *input* variable i.

The objective of the commands, called *output* variable o and sent to the actuators, is to modify the value of some characteristics of the environment, called a *controlled* variable c. Variables m and c are called *environment variables*. Variables i and o are called *controller variables*. Finally, a controller has its own internal state variables to perform computations. In this case study, we use Event-B state variables to represent both environment and controller variables. We model neither sensor/actuator failures nor their delays.

A well-known architecture of a control system is a control loop that reads all input variables at once, at a given moment, and then computes all output variables in the same iteration. But, it can be also viewed as a continuous system that can be interrupted by any change in the environment represented by a new value sent by a sensor. In this paper, we adopt a hybrid view: every 10 seconds, the AMAN reads various sensor inputs and makes a new prediction to display. Moreover, the AMAN instantaneously reacts to some ATCo's requests by updating the display. The use of the four-variable model permits to cope with the complexity of such systems and define a generic methodology for modeling them using Event-B. This methodology consists in defining a

separate event for each input received by the system or output sent towards the environment. Therefore, modelling the AMAN with Event-B consists in defining one event for each input corresponding to the ATCo's interaction with its environment and an additional event display representing the output of the system, that is, calculating and displaying a new prediction performed by the AMAN.

2.2 Modeling structure

The Event-B specification presented in this paper is incrementally built using refinement. It is composed of seven levels (seven machines and two contexts) and defines and uses a theory to deal with, among others, sequences, the absolute value, etc. (see Figure 2).

- Context C1 mainly defines the following constants: *Labels* to represent the aircraft, *Hours*, *Minutes* and *Seconds* to denote the possible values of these time units, *zoomLevels* representing the possible values for the zoom level.
- Machine M1 sees the context C1 and defines, among others, an event for determining and displaying the arrival times of aircraft and an event for selecting the zoom level.
- Machine M2 models the holding of an aircraft. This machine sees the context C1.
- Machines M3 and M4 respectively introduce the moving of a scheduled aircraft to change its arrival time and the blocking of time slots by the ATCo. Both machines M3 and M4 see the context C1.
- Machine M5 represents the request of an aircraft for landing. It sees the context C1. In each of the machines (M1-M5), we model the historical functions that permit the AMAN to provide the ATCo with the previous predictions.
- Machine M6 models the interaction between the ATCo and the AMAN using the mouse. This machine sees the context C2 that defines some constants to describe a mouse in terms of its possible states (*clicked*, *released*) and also the different elements on which a mouse may click.
- Machine M7 specifies the AMAN breakdown for failure for instance. It sees the context C2.

Roughly speaking, the structure of the Event-B specification is constructed as follows: the outputs (AMAN's functionalities, that are, the prediction and display of the arrival times) are modeled first in the machine M1, then the ATCo's interactions are modelled in a second step (Machines M2-M6). Finally, the last level M7 models the failure of the AMAN.

2.3 Formalisation of the requirements and tasks

Table 1 shows where and how the requirements, listed in [25] and summarized in Appendix A, are specified in our Event-B models. As one can remark, depending on the kind of the requirement, this later is specified as an invariant, a guard, an elementary variable (like the variable *mouseState*), an event with specific guards, etc. Requirements **Req22** and **Req23** for instance cannot be easily expressed as an invariant since it would require to introduce at least three additional variables: (i) a variable *mouseStateP* to store the previous state of the mouse, (ii) a variable *mousePositionP* to store the previous position of the mouse and (iii) a variable *isEnabled* to know whether the hold button is enabled or not. In that case, the invariant would be expressed as follows:

$$\begin{array}{l} mouseStateP{=}clicked \wedge mousePositionP{=}hold \\ mouseState{=}released \wedge mousePosition{=}hold \\ \Rightarrow \\ isEnabled = \textbf{TRUE} \end{array}$$

We did not choose this option because these additional variables make the specification more complex; we have to manage their updates by each event. Finally, let us note that some requirements (Requirements **Req17-18** and **Req20**) are not covered because they are related to the interface appearance and not to the system functionalities. Indeed, the Event-B method is dedicated to specifying the state of a system and its evolution and not to describing the visual appearance of its components. In addition to the requirements listed in Table 1, we have specified some additional properties that we consider of good sense. For instance, we have specified that the requests are dealt with according to the FIFO strategy (First In First Out) to ensure fairness. More details are given in the next section.

Table 2 shows the correspondence between the ATCo's actions/AMAN tasks and the events of the built specification.

3 Description of the Event-B Models

In this section, we give a brief description of some key refinement levels of the Event-B modeling of the AMAN. The complete archive of the Event-B project is available in [23]. Our modeling makes the assumption that the AMAN predictions are done for a single day, that is, no aircraft is planned for the next day.

3.1 Machine M1

Machine M1 models the prediction of the arrival times of aircraft (called labels in the rest of the paper) and its display on the screen. This machine sees the context C1 that defines: the set of all possible objects of the system (*Elements*), the set of possible labels (*Labels*), the units of time (*Seconds*, *Minutes* and *Hours*), the zoom levels (*zoomLevels*), the time progression step (*step*) and the security distance between two distinct labels (*sep*):

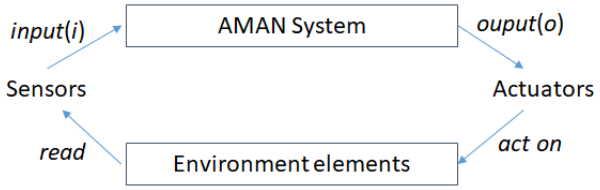

Fig. 1. Four-variable model

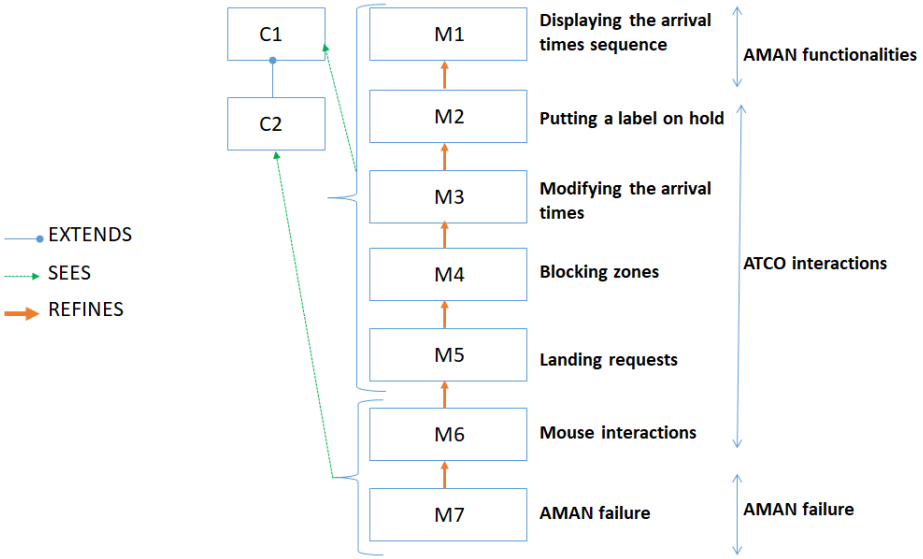

Fig. 2. Event-B structure of the project

Requirements	**Components**	Event-B **element**
Req1	Machine M5	Invariant inv1
Req2	Machine M2	Invariant inv1
Req3	Machine M3	Invariant inv6
Req4	Machine M2	Event removeholdLabel
Req5	Machines M1 and M3	Invariants inv10(M1) and inv5(M3)
Req6	Machine M4	Invariants inv1 and inv2
Req7	Machine M3	Invariant inv6
Req8	Machine M7	At any moment, the Boolean Variable *isStopped* can be updated by the event stopStart. All other events are guarded by (*isStopped*= **FALSE**)
Req9 to **Req13**	Machines M1-M6	HTimed variables V_T
Req14		Not covered
Req15	Machine M6	Event holdLabel is enabled only if a label is selected (guard grd2)
Req16	Context C1	Axiom axm6
Req17 **Req18**		Not covered
Req19	Machine M1	Invariant inv12
Req20		Not covered
Req21	Machine M6	Variable *mouseState*
Req22 **Req23**	Machine M6	Guard grd4 of the event holdLabel

Table 1. Cross-reference between the components of our model and the requirements of [25]

Tasks	Components	EVENT-B element
Compute/Display LS	M1	Event Display
Change zoom	M1	Event selectZoom
Put Aircraft on hold	M2	Events holdLabel and removeholdLabel
Change LS	M3	Event moveLabel
Block time slot	M4	Events addBlockedZone and deleteBlockedZone
Stop Manage LS	M7	Event stopStart

Table 2. Cross-reference between the components of our model and the ATCo's actions/AMAN tasks of [25]

SETS *Elements*
CONSTANTS *Labels*, *Seconds*, *Minutes*, *Hours*, *zoomLevels*, *step*, *sep*
AXIOMS
axm1: **finite**(*Elements*) ∧ *Labels* ⊆ *Elements*
axm2: *Seconds* = 0..59 ∧ *Minutes* = 0..59 ∧ *Hours* = 0..23
axm3: zoomLevels ⊆ ℕ1 ∧ *zoomLevels* ≠ ∅
axm4: *step* ∈ ℕ1 ∧ *sep* ∈ ℕ1

The machine M1 defines the following invariants to characterise the possible arrival times of a set of labels where *curTimeMin* (resp. *curTimeSec*) gives the time in terms of minutes (resp. seconds):

inv1 $labels\_T \in (\bigcup k.\ k \in \mathbb{N} \wedge step \times k \leq curTimeSec(curTimeS, curTimeM, curTimeH) \mid step \times k) \rightarrow \mathbb{P}(Labels)$
inv2 $arrivalM\_T \in (\bigcup k.\ k \in \mathbb{N} \wedge step \times k \leq curTimeSec(curTimeS, curTimeM, curTimeH) \mid step \times k) \rightarrow (Labels \nrightarrow Minutes)$
inv3 $arrivalH\_T \in \mathbf{dom}(arrivalM\_T) \rightarrow (Labels \nrightarrow Hours)$
inv4 $\forall t \cdot t \in \mathbf{dom}(arrivalM\_T) \Rightarrow \mathbf{dom}(arrivalM\_T(t)) \subseteq labels\_T(t)$
inv5: $\forall t, l.\ t \in \mathbf{dom}(arrivalM\_T) \wedge l \in \mathbf{dom}(arrivalM\_T(t)) \Rightarrow curTimeMin((arrivalM\_T(t))(l), (arrivalH\_T(t))(l)) \geq t \div 60$
inv6: $\forall t, l1, l2 \cdot t \in \mathbf{dom}(arrivalH\_T) \wedge l1 \in \mathbf{dom}(arrivalM\_T(t)) \wedge l2 \in \mathbf{dom}(arrivalM\_T(t)) \wedge l1 \neq l2 \Rightarrow \mathbf{abs}(curTimeMin((arrivalM\_T(t))(l1), (arrivalH\_T(t))(l1)), curTimeMin((arrivalM\_T(t))(l2), (arrivalH\_T(t))(l2))) \geq sep$
inv7: $zoomLevel \in zoomLevels \wedge displayedLabels \subseteq \mathbf{dom}(arrivalM\_T(t))$
inv8: $displayedLabels = (\bigcup l.\ l \in \mathbf{dom}(arrivalM\_T(t)) \wedge curTimeMin((arrivalM\_T(t))(l), (arrivalH\_T(t))(l)) \leq curTimeMin(curTimeM, curTimeH) + zoomLevel \mid \{l\})$

Invariant inv1 defines the set of existing labels at each moment. Invariants inv2, inv3 and inv4 describe the arrival times of a set of labels. Invariant inv5 states that the arrival time of an aircraft is later than the current time while inv6 ensures the security of passengers by separating the labels by at least *sep* minutes. Finally, the invariants inv7 and inv8 specify the set of labels displayed on the screen according to their arrival times and the selected zoom *zoomLevel*. Variable *zoomLevel* is an integer (between 15 and 45) that defines the display window of the labels: a label is displayed if its arrival time falls into this window (inv8). We have chosen to model the zoom functionality and the calculation of the label arrival times at the same level because both modify the variable *displayedLabels*. As stated in Section 1.1, all events modifying a variable must be specified in the same machine where the variable is defined. Another option would be to define the variables *displayedLabels* and *zoomLevel* in an other refinement level. We did not choose this option because it adds a refinement level while including them in M1 does not add any complexity.

Machine M1 defines the event display as follows:

Event display $\widehat{=}$
any
t, *landingLabs*, *labsToDisp*, *labsSch*, *arr*, *ns*, *nm*, *nh*
where
grd1: $t = curTimeSec(curTimeS, curTimeM, curTimeH) + step$
grd2: $landingLabs = (\bigcup l.\ l \in \mathbf{dom}(arrivalM\_T(t\text{-}step)) \wedge curTimeMin(nm, nh) > curTimeMin((arrivalM\_T(t\text{-}step))(l), (arrivalM\_T(t\text{-}step))(l)) \mid \{l\})$
grd3: $labsSch \subseteq labels\_T(t\text{-}step) \setminus landingLabs$
grd4: $arr \in labsSch \rightarrow curTime(nm, nh)..curTime(nm, nh)+180$
grd5: $\forall l1, l2 \cdot l1 \in \mathbf{dom}(arr) \wedge l2 \in \mathbf{dom}(arr) \wedge l1 \neq l2 \Rightarrow \mathbf{abs}(arr(l1), arr(l2)) \geq sep$
grd6: $toDisp = (\bigcup l \cdot l \in \mathbf{dom}(arr) \wedge arr(l) \leq curTimeMin(nm, nh) + zoomLevel \mid \{l\})$
grd7: ...
then
act1: $displayedLabels := toDisp$
act2: $arrivalM\_T(t) := (\lambda\ l.\ l \in \mathbf{dom}(arr) \mid arr(l)\ \mathbf{mod}\ 60)$
act3: $arrivalH\_T(t) := (\lambda\ l.\ l \in \mathbf{dom}(arr) \mid arr(l) \div 60)$
act4: $curTimeS := ns$
act5: $curTimeM := nm$
act6: $curTimeH := nh$
act7: $labels\_T(t) := labels\_T(t\text{-}step)$
end

where *ns* (resp. *nm*, *mh*) denotes the second (resp. minute, hour) unit of the current time plus 10 seconds. Roughly speaking, this event starts by calculating the set of labels that have already landed (Guard grd2), then it makes a prediction for a subset *labsSch* of other existing labels (Guards grd3 and grd4) by ensuring that the labels are separated by at least *sep* minutes (Guard grd5). Finally, it calculates the set of the labels to display according to their arrival times and the selected zoom level (Guard grd6). Actions update different variables by stamping them with the current time *t*.

Let us remark that the guard grd4 specifies that the predictions are made for the next 3 hours. We put this hypothesis in order to improve the PROB performance and make the animation of the models possible. Accord-

ing to the case study authors, such an assumption is very reasonable and is not a limitation of the model.

3.2 Machine M2

Machine M2 models labels put on hold. For that purpose, the following invariants are defined. Invariant inv1 types the introduced variable *holdLabels_ T*. Invariant inv2 states that a label made on hold must be removed from the arrival sequence of the next cycle (*x+step*).

inv1: $holdLabels\_\ T \in$
$(\bigcup k.\ k \in \mathbb{N} \wedge step \times k \leq curTimeSec(curTimeS, curTimeM, curTimeH) \mid step \times k) \rightarrow \mathbb{P}(\text{Labels})$
inv2 $\forall x \cdot x \in \mathbf{dom}(holdLabels\_\ T) \wedge x + step \in \mathbf{dom}(arrivalM\_\ T) \Rightarrow holdLabels\_\ T(x) \cap \mathbf{dom}(arrivalM\_\ T(x + step)) = \varnothing$

Machine M2 defines an event holdLabel that permits to put on hold a displayed label *l* by adding it to *holdLabels*.

Event holdLabel $\widehat{=}$
any
$t\ l$
where
grd1: $t = curTimeSec(curTimeS, curTimeM, curTimeH)$
grd2: $l \in \mathbf{dom}(arrivalM\_\ T(t)) \setminus holdLabels\_\ T(t)$
then
act1: $holdLabels\_\ T(t) := holdLabels\_\ T(t) \cup \{l\}$
end

The event display is refined by adding the guard ($labsSch \cap holdLabels\_\ T(t - step) = \varnothing$) in order to maintain the invariant inv2 by removing the held labels from the arrival sequence.

3.3 Machine M3

Machine M3 models the request of the ATCo that would like to change the arrival time of a label by defining the following invariants. Invariants inv1 and inv2 define the new arrival times of a given label. Invariant inv3 states that only scheduled labels, which are not put on hold, can be moved and new arrival times are suggested by the ATCo. Invariant inv4 models the requirement **Req5** to avoid overlapping labels.

inv1: $newArrivalM\_\ T \in$
$(\bigcup k.\ k \in \mathbb{N} \wedge step \times k \leq curTimeSec(curTimeS, curTimeM, curTimeH) \mid step \times k) \rightarrow (Labels \nrightarrow Minutes)$
inv2: $newArrivalM\_\ T \in \mathbf{dom}(newArrivalM\_\ T) \rightarrow (Labels \nrightarrow Hours)$
inv3: $\forall x \cdot x \in \mathbf{dom}(newArrivalM\_\ T) \Rightarrow \mathbf{dom}(newArrivalM\_\ T(x)) \subseteq \mathbf{sdom}(arrivalM\_\ T(x)) \setminus holdLabels\_\ T(x)$
inv4 $\forall t, x, y \cdot t \in \mathbf{dom}(newArrivalM\_\ T) \wedge x \in \mathbf{dom}(newArrivalM\_\ T(t)) \wedge y \in \mathbf{dom}(newArrivalM\_\ T(t)) \wedge x \neq y \Rightarrow (newArrivalM\_\ T(t) \otimes newArrivalH\_\ T(t))(x) \neq (newArrivalM\_\ T(t) \otimes newArrivalH\_\ T(t))(y)$

As stated in the requirement document, a moving request might be rejected by the AMAN if it would require a speed-up of the aircraft beyond the capacity of the aircraft. To model such a requirement, we added the following guards to the event display that specify that the labels that cannot be moved must keep their original arrival times (guard grd2), whereas others are moved to the new ones (guard grd3). Guard grd4 permits to cover the requirement **Req7** by stating that the labels that ask for delaying its arrival time should be satisfied. Function *canBeMoved* permits to abstract from the details and calculations made by the AMAN to decide whether a label can be moved or not. Such details can be introduced later by refining this function. As the requirement document does not give enough information about this point, we kept the function *canBeMoved* in its abstract form.

grd1: $canBeMoved \in \mathbf{dom}(newArrivalM\_\ T(t - step)) \setminus landingLabs \rightarrow \mathbf{BOOL}$
grd2: $canBeMoved^{-1}[\{\mathbf{TRUE}\}] \lhd arr = (\lambda x.\ x \in canBeMoved^{-1}[\{\mathbf{TRUE}\}] \mid curTimeMin\ (\ (newArrivalM\_\ T(t - step))\ (x), (newArrivalH\_\ T(t - step))(x)))$
grd3: $canBeMoved^{-1}[\{\mathbf{FALSE}\}] \lhd arr = (\lambda x.\ x \in canBeMoved^{-1}[\{\mathbf{FALSE}\}] \mid curTimeMin\ ((newArrivalM\_\ T(t - step))(x), (newArrivalH\_\ T(t - step))(x)))$
grd4: $(\bigcup x \cdot x \in \mathbf{dom}((newArrivalM\_\ T(t - step))) \setminus landingLabs \wedge curTimeMin((newArrivalM\_\ T(t - step))(x), ((newArrivalH\_\ T(t - step))(x)) > curTimeMin((newArrivalM\_\ T(t - step))(x), (newArrivalH\_\ T(t - step))(x)))|\{x\}) \subseteq canBeMoved^{-1}[\{\mathbf{TRUE}\}]$

Machine M3 is refined by the machine M4 to model the blocked slots. As its Event-B modeling is very similar to that of held labels, this paper does not give more details about the machine M4.

3.4 Machine M5

The machine M5 models the flights approaching an airport as an injective sequence of requests submitted to the AMAN (Invariants inv1 and inv2). For that purpose, we have specified a theory to define the sequence data structure along with its associated operations like inserting/deleting elements.

inv1: $requests\_\ T \in (\bigcup k.\ k \in \mathbb{N} \wedge step \times k \leq curTimeSec(curTimeS, curTimeM, curTimeH) \mid step \times k) \rightarrow \mathbf{seq}(Labels)$
inv2: $\forall t, x, y.\ t \in \mathbf{dom}(requests\_\ T) \wedge x \in \mathbf{dom}(requests\_\ T(t)) \wedge y \in \mathbf{dom}(requests\_\ T(t)) \wedge x \neq y \Rightarrow (requests\_\ T(t))(x) \neq (requests\_\ T(t))(y)$

Moreover, to ensure that the requests are fairly processed, that is, the label that made a request in first will get an arrival time earlier than others (FIFO strategy), we have defined the following invariant:

inv3: $\forall\, t, l1, l2 \cdot\ t \in \mathbf{dom}(requests\_\ T) \wedge$
$t + step \in \mathbf{dom}(arrivalM\_\ T) \wedge l1 \in \mathbf{ran}(requests\_\ T(t)) \wedge$
$l2 \in \mathbf{ran}(requests\_\ T(t)) \wedge$
$(requests\_\ T(t))^{-1}(l1) < (requests\_\ T(t))^{-1}(l2)$
$\Rightarrow$
$curTimeMin((arrivalM\_\ T(t+step))(l1),$
$(arrivalH\_\ T(t + step))(l1))$
$<$
$curTimeMin((arrivalM\_\ T(t+step))(l2),$
$(arrivalH\_\ T(t + step))(l2))$

Machine M5 also defines an event to add/delete requests. Moreover, we have added the following guards to the event display: the guard grd1 states that the AMAN should predict arrival times for the labels having made requests and the already scheduled labels that are not made on hold or landed. The guard grd2 specifies the FIFO strategy for request processing.

grd1: $labsSch = (\mathbf{ran}(requests\_\ T(t - step)) \cup$
$\mathbf{dom}(arrivalM\_\ T(t - step))) \setminus$
$(landingLabs \cup holdLabels\_\ T(t - step))$
grd2: $\forall\, x, y \cdot\ x \in \mathbf{dom}(requests\_\ T(t - step)) \wedge$
$y \in \mathbf{dom}(requests\_\ T(t - step)) \wedge x > y$
$\Rightarrow$
$arr((requests\_\ T(t - step))(x)) >$
$arr((requests\_\ T(t - step))(y))$

3.5 Machine M6

This machine models the interactions of the ATCo with the AMAN using the mouse. Context C1 is extended by the context C2 that defines the set *Elements* as a partition of *Labels*, the button *hold*, the slide-bar *zoom* for changing the zoom and *nothing* to model the mouse that clicks on any other zone of the interface. A set representing the possible states of the mouse is also defined:

ax1: $\mathbf{partition}(mouseStates, \{released\}, \{clicked\})$
ax2: $\mathbf{partition}(Elements, Labels, \{nothing\}, \{hold\}, \{zoom\})$

In the machine M6, we introduced two additional variables *clickedElement* and *selectedElement* to respectively denote the element the mouse clicks on or selects. Both variables belong to $((displayedLabels \cup \{nothing, hold, zoom\}) \setminus holdLabels)$. We also describe a set of events to model the behavior of the mouse like clicking on or selecting an element. Event holdLabel is refined by:

Event holdLabel $\widehat{=}$
refines holdLabel
where
grd1: $selectedElement \in \mathbf{dom}(arrivalM\_\ T(t)) \setminus holdLabels\_\ T(t))$
grd2: $selectedElement \notin \mathbf{dom}(newArrivalM\_\ T(t))$
grd3: $mousePosition = hold \wedge mouseState = clicked$
with
1: $l = selectedElement$
then
act1: $holdLabels\_\ T(t) := holdLabels\_\ T(t) \cup \{selectedElement\}$
act2: $selectedElement := nothing$
act3: $clickedElement := nothing$
act4: $mouseState := released$
end

The refinement of the event holdLabel states that the label l to hold denotes the label selected by the mouse (the with clause). To put the selected label on hold, the guard grd3 specifies that the mouse must be in the state *clicked* and positioned on the hold button.

4 Validation and Verification

To verify the correctness of our models and ensure that we built the right system, we have performed the four steps detailed hereafter.

4.1 Model checking of the specification

As one can remark, some refinement levels contain invariants that depend on several variables. In that case, it becomes quite difficult to find the right specification (guards/actions) the first time. The ProB model checker has proven very useful in finding actions/guards to add to some events in order to establish these invariants. Basically, before performing the proof that may be tedious, we used the ProB model checker to exhibit some possible scenarios that violate the invariant. A scenario is a sequence of events that, starting from a valid initial state of the machine, leads to a state that violates the related invariant. Analysing such scenarios helps us correct the specification by adding guards/actions to some events but also sometimes to revise the invariants. For this particular case study, the use of ProB helped us particularly to find the following elements:

- the definition of the term $(\bigcup k.\ k \in \mathbb{N} \wedge step \times k \leq curTimeSec(curTimeS, curTimeM, curTimeH) \mid step \times k)$ used to type the historical variables: inv1 of the machine M1 for instance,
- the action act7 on the variable $labels_T(t)$ in order to maintain the history of the existing labels;
- the guards grd2-grd4 added to the event disply in the machine M3.

4.2 Proof of the specification

Even if ProB does not find any scenario that violates the invariant, this does not mean that the models are correct. Indeed, by default ProB limits the number of explored parameter values per event. Some of the events like `addRequest` have a very large number of possible valuations of the parameters, which prevents the model checker from exploring the full state space and exploring certain scenarios. It is, however, possible in the animator interface of ProB to provide explicit values for the event parameters and thus check given scenarios by trace replay. Still, proof is indispensable for exhaustive verification. Proof in Event-B aims at verifying that each event does preserve the invariant and that the guard of each refined event is stronger than that of the abstract

Element Name	Tot.	Aut.	Man.	Rev.	Und.
STTT_AMAN	**329**	**178**	**151**	**0**	**0**
C1	12	8	4	0	0
C2	1	1	0	0	0
C3	1	1	0	0	0
M1	74	44	30	0	0
M2	29	18	11	0	0
M3	53	25	28	0	0
M4	46	22	24	0	0
M5	60	25	35	0	0
M6	53	34	19	0	0
M7	0	0	0	0	0
M8	0	0	0	0	0

Tot.: Total ; Aut.: Automatic; Man.: Manual; Rev.: Reviewed; Und.: Undischarged.

Fig. 3. RODIN proof statistics of the case study

one. The corresponding proof obligations are automatically generated by RODIN. Figure 3 provides the proof statistics of the case study: 329 proof obligations have been generated, of which 54% (178) were automatically proved by the various provers[3]. The interactive proof of the remaining proof obligations (manual ones) took about one week since they are more complex (in particular those that depend on the historical variables) and require several inference steps and need the use of external provers (like the Mono Lemma prover, *Dis-prove* with PROB and STM provers). During an interactive proof, users ask the internal prover to follow specific steps to discharge a proof obligation. A step proof consists in applying a deductive rule, adding a new hypothesis that is in turn proved or calling external provers. The external Mono Lemma prover has been very useful for arithmetic formulas, even if we had to add the following theorem on the modular operator:

$$\forall\, x, y.\ x \in \mathbb{N} \wedge y \in \mathbb{N}1 \Rightarrow x = x \bmod y + x \div y \times y$$

It is worth noting that performing interactive proofs does not decrease the confidence of the models since the proofs are discharged under the RODIN platform by enriching the prover only by theorems that are proved as well.

4.3 Validation with scenarios

Defining and simulating scenarios enables us to validate whether we have built models that behave as expected. Unfortunately, the requirement document does not provide scenarios that would help us in such a task. Therefore, we have defined our own scenarios based on our understanding of the system. Basically, we have defined a validation scenario for each AMAN functionality and ATCO interaction. From a practical point of view, for each invariant modeling a property, we define a scenario to check whether the built models fulfill the related property. Using the animation capability of PROB, we have checked, among others, the following behaviours:

- moving a label l_1 to a slot that does not respect the 3 minutes separation with an other label l_2: in that case, the AMAN also moves l_2 and its neighbours to maintain this security requirement (Invariant inv6 of the machine M1).
- putting a label on hold results in removing it from the landing sequence: such an effect is not instantaneous; it is performed by the AMAN in the next processing cycle (Invariant inv2 of the machine M2).
- blocking a slot time results in moving all the labels scheduled in this slot into other available slots.
- the landing requests received by the AMAN are dealt with according to the FIFO strategy. Moreover, the corresponding labels are scheduled after the already scheduled ones (Invariant inv3 of the machine M5).
- selecting a zoom level does display only the labels that are scheduled in the corresponding slot (next (current time + zoom) minutes). Contrary to the previous scenarios, the effect of this ATCO action is instantaneous (Invariant inv8 of the machine M1).

4.4 Validation with Visualisation

We used visualisation to better debug counter examples and scenarios, but also during interactive animation. Our visualisation of the model was achieved using the VISB [30] component of PROB. The visualisation uses a SVG graphics file and a JSON glue file. The latter contains a mapping between the B model and the graphics file, and can add, show, hide, move and alter elements from the SVG file.

To develop our visualisation we built on the VISB files developed for another Event-B model [15] of the AMAN case study. Even though this Event-B model was developed independently and has a quite different structure (cf. Section 5 below), we managed to reuse the SVG file and adapt the JSON glue file. This was possible thanks to several extensions to the VISB JSON glue file format, not available in the first version [30]: e.g., allowing to add objects to the SVG file or allowing to provide local definitions visible within VISB expressions (in addition to variables and constants coming from the model). Here we describe the core contents of our VISB JSON glue file:

- a pointer to the SVG file. In our case this is the SVG file developed for [15]. This SVG file contains many objects visible in Fig 4, such as the rectangle `bt_hold` or the text field `tx_hold`:

```
<rect id="bt_hold" fill="black"
```

[3] The theory gives 24 additional proof obligations, of which 23 have been interactively proved. Unfortunately, RODIN does not include the proof obligations related to theories in the statistics of the project.

```
    visibility="visible" width="120"
    height="30" x="650" y="500"/>
  <text id="tx_hold" stroke="red"
    fill="red" visibility="true"
    x="688" y="520">HOLD</text>
```

– a list of additional SVG objects added to the ones in the SVG file. In our case, we added a text field to display the status of the mouse.
– a list of local definitions. For example, the definitions header contains this entry, mimicking the set of known and visible planes as well as a boolean variable from [15]:

```
"definitions": [
  { "name":"known_planes",
    "value" : "labels"
  },
  { "name":"visible_planes",
    "value" : "diplayedLabels"
  },
  { "name": "no_airplane_is_selected",
    "value": "bool(selectedElement=nothing)"
  },
```

Other defined the `Airplanes` constant in [15], deriving it from Elements:

```
  { "name": "Airplanes",
    "value": "Elements - {nothing,hold,zoom}",
    "comment": "Remove special elements"
  },
```

These three definitions computes explicitly the current landing sequence:

```
  { "name": "arrivalH",
    "value": "arrivalH_T(curTimeSec(curTimeS,
                        curTimeM,curTimeH))"
  },
  { "name": "arrivalM",
    "value": "arrivalM_T(curTimeSec(curTimeS,
                        curTimeM,curTimeH))"
  },
  { "name": "landing_sequence",
    "value": "%plane.(plane:dom(arrivalH) |
    arrivalH(plane)*60+arrivalM(plane) -
    currentTime+1)"
  },
```

– a list of updates, called items.
The following VisB item uses the above definition to set the visibility attribute of the hold button (visible in Figure 4).

```
  {
    "id": "bt_hold",
    "attr": "visibility",
    "value":"IF no_airplane_is_selected=TRUE
             THEN \"hidden\"
             ELSE \"visible\"
             END"
  },
```

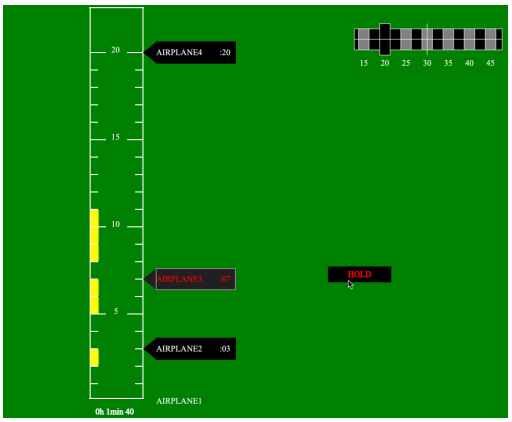

Fig. 4. Visualising the state of the model (M8) using ProB and VisB

Observe that the value is a classical B expression, which has access to ProB's IF-THEN-ELSE for expressions and data types such as strings and reals, as well as ProB's external B functions to manipulate them.
– a list of events.
The VisB file also associates the event `holdLabel` with the hold button; i.e., it is executed when the button is clicked. A hover is also specified, that updates the image when the mouse hovers above the button.

```
  {
    "id": "bt_hold",
    "event": "holdLabel",
    "hovers" : [{"id":"bt_hold","attr":"fill",
    "enter":"gray", "leave":"black"}]
  },
```

Observe, that Fig 4 also shows a mouse pointer over the hold button: this is also part of the visualisation and is based on the `mousePosition` variable of the model. Another interesting aspect is that validation traces can be exported to standalone HTML files using ProB (see [29]). These traces can be reused to step through the traces and inspect the visualisation and the variable values, without access to either ProB, Rodin or the Event-B model. We used those HTML trace files as a means of (email) exchange between ourselves, e.g., to point out and discuss tricky aspects of the models. (Some of these traces are available at `https://stups.hhu-hosting.de/models/AMAN/` and [23] for reference.).

5 Discussion and Comparison

In a companion paper [15] an Event-B model was developed independently.[4] The models have very different

[4] The model of this paper was developed by the first author; the second author only intervened in the validation and verification, not in the writing of this model.

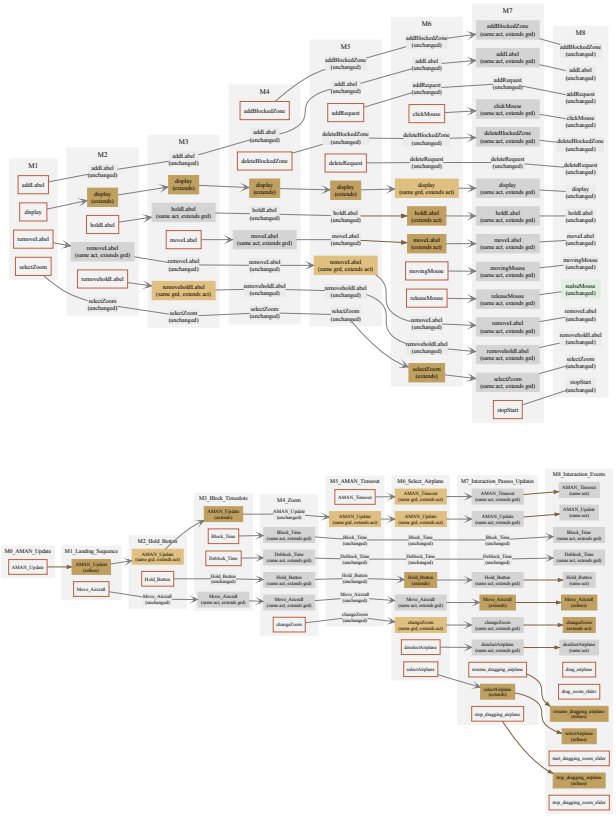

Fig. 5. The event refinement hierarchies of our model (top) and of the companion model [15] bottom (generated by ProB)

refinement strategies, as can be guessed from looking at Figure 5. The figure shows the event refinement hierarchy: on the left you can see all events of the most abstract model, while the most concrete model is depicted on the right. Newly introduced events (refining skip) are shown with a red border, while unchanged events are shown in light gray and are not connected with an arrow (just a straight line). Extended events are shown in brown. Our most abstract model (on the top left in Fig. 5) already has four events and twelve variables modeling the concepts of labels and zooming, while the most abstract model of [15] has just one event and variable and zooming is only added in machine M4.

Note that [15] has two more refinement levels not shown in Fig. 5: M9 introducing mouse movements and M10 adding precise pixel locations (something which we did not model in this article). The models also have a quite different set of variables: M8 of our model at ABZ'2023 had 32 variables, the current version has merged some of them and now has 18 variables (see Fig. 6), the comparable M9 of [15] has 17 variables). Still, as seen above, we were able to reuse its VisB visualisation for our model.

Our development models the current time (as three variables for hours, minutes and seconds), which increases during execution (of the autonomous AMAN event dis-

Fig. 6. The variable refinement hierarchies of our model (generated by ProB)

play). This is better for simulation, but more tricky for model checking as the full state space is automatically infinite (unless we restrict model checking to a particular time interval). In the companion model, the authors normalize the current time to 0; in other words, all times are relative to the current time (cf. [28] or [16]). In addition, the other model only models time as relative minutes, abstracting away both hours and seconds. The use of relative time makes state space finite, but for deeper levels it is too large without other restrictions as well (there are 2^{45} values already just for blocking time slots).

In this article, we have used the theory plugin [8] to deal with sequences and with time. For sequences we use the standard Rodin theory, providing direct definitions for sequence operators (like `seqAppend` to concatenate two sequences). The Time theory provides operators to manipulate times and time differences expressed in minutes, hours and seconds. The theory also contains an absolute value function, which is encoded in a context in [15] (which is possible, as the function is not polymorphic). The sequence operations were encoded "by hand" in [15] using relational operators.

In this model, we have also three special elements `hold`, `nothing` and `zoom`, while the other model only has airplane elements and encodes the special values via sets. E.g., `selectedElement=nothing` corresponds to `selectedAirplane` $= \varnothing$ in [15]. The set encoding requires an additional invariant `card(selectedAirplane)` $\leq$ `1` in [15]. On the one hand, this makes it easier to adapt the model to allow selecting multiple airplanes later, but, on the other hand induces a few more well-definedness POs (due to the use of `card`), which were

surprisingly tedious to discharge. We did not notice any fundamental differences otherwise. It is thus worth noting that the models represent two alternative Event-B specifications of the same system.

In [5], the model-view-controller (MVC) pattern [1] has been extended to integrate ASM formal specifications for the development of user interfaces (UI). Mainly, the approach is composed of three steps: (1) an MVC pattern is specified to model the interface of the system and users; (2) the behavior of the considered system is then formally specified using ASM notations [6]; in the last step (3) the MVC elements which represent the user interaction with the system are linked to ASM elements using Java annotations. This is very similar to the use of VisB with Event-B models for the purpose of animation. As for Event-B, properties are specified as invariants and guaranteed by attaching guards to ASM actions that make them enabled only if these guards are fulfilled. For the purpose of verification, some safety properties are modelled as LTL formulas and then verified using the NuSMV model checker [9]. We think that some of them can be simply modelled as invariants since they depend on variables whose values are taken in the same state. The main drawback of this approach is that all the properties are specified at the first refinement level; this makes it cumbersome.

In [13], the Alloy formal language is used to model and verify the AMAN system. The authors propose first a generic specification of the HAMSTERS task model (structural and semantics aspects)[4] in Alloy, then this latter is instantiated on the AMAN system in order to verify properties on the user interactions with the system. This approach also considers the robustness of the AMAN system against ATCo errors represented as unexpected ATCo actions. Properties to verify are specified as Alloy assertions that are then checked using the bounded model checking engine of Alloy 6. The verification phase emphasises on the user interaction rather than the internal functionalities of the AMAN system. Consequently, this approach does not consider all the requirements reported in [25].

The approaches presented in [5,13] both use bounded model-checkers for verification. To avoid the well-known problem of state space explosion related to such tools, the properties verification is only performed for a bounded number of objects (labels, slots, etc.).

6 Conclusion

In this paper, we presented an Event-B formal model of the Arrival MANager system (called AMAN), the case study provided in the ABZ2023 conference. Our specification takes most requirements into account and defines additional ones that can be considered common sense, like a fair processing of landing requests. Compared to previous case studies proposed in the ABZ conferences, this present case study contains fewer invariants (65 invariants) but most of them are dynamic and require thus the introduction of several auxiliary variables to store the previous system state. This implies the definition of additional invariants to relate the before and current values of each variable. These additional invariants produce a great number of proof obligations since we have to establish that each event maintains these invariants.

For this particular case study, the expression of invariants that depend on the previous state proved to be difficult since variables are interrelated: at the instant t, the arrival times of aircraft depend on the moved labels, requests and aircraft put on hold during the last 10 seconds (at the instant (t-*step*)). The use of ProB helped us in defining the correct expression of these invariants by model checking invariants and simulating some scenarios to validate/fix them. The user-friendly graphical visualisation makes the validation phase easier.

Compared to the previous ABZ case studies [21,24], the present case study is time-dependent. Indeed, its main objective is to assign arrival times to aircraft; this is why we introduced timed aspect from the first specification level along with the event display that makes the time evolve.

In future work, we plan to study and model how AMAN can decide whether a label can be moved or not. For this purpose, we can assume that an arrival time interval is associated with each label. In that case, AMAN would allow the moving of a label *iff* it remains within its associated interval. Unfortunately, we fell short of time to deeply investigate this solution. Future improvements also include exploring the use of decomposition plugins available in Rodin for structuring the built models into smaller and thus more manageable units. We can see the system as a set of independent parts (each of them corresponding to a single ATCo interaction) and the AMAN as a root part that uses their information to calculate a new prediction.

Acknowledgements We would like to thank Fabian Vu for developing the VisB visualisation of the companion model. We thank David Geleßus and Sebastian Stock for fruitful discussions about the case study. We also thank anonymous referees for their useful feedback.

References

1. Model-View-Controller Pattern, pp. 353–402. Apress, Berkeley, CA (2009), https://doi.org/10.1007/978-1-4302-2370-2_20
2. Abrial, J.R.: The B-Book - Assigning Programs to Meanings. Cambridge University Press (1996)
3. Abrial, J.: Modeling in Event-B. Cambridge University Press (2010)
4. Amor, M.B.: Hamsters: A New Task Model for Interactive Systems (2009)

5. Bombarda, A., Bonfanti, S., Gargantini, A.: Formal MVC: A Pattern for the Integration of ASM Specifications in UI Development. In: Glässer, U., Campos, J.C., Méry, D., Palanque, P.A. (eds.) Rigorous State-Based Methods - 9th International Conference, ABZ 2023, Nancy, France, May 30 - June 2, 2023, Proceedings. Lecture Notes in Computer Science, vol. 14010, pp. 340–357. Springer (2023)
6. Börger, E., Stärk, R.F.: Abstract State Machines. A Method for High-Level System Design and Analysis. Springer (2003)
7. Butler, M., Maamria, I.: Mathematical Extension in Event-B through the Rodin Theory Component (2010)
8. Butler, M.J., Maamria, I.: Practical Theory Extension in Event-B. In: Theories of Programming and Formal Methods - Essays Dedicated to Jifeng He on the Occasion of His 70th Birthday. pp. 67–81 (2013)
9. Cimatti, A., Clarke, E.M., Giunchiglia, F., Roveri, M.: NUSMV: A New Symbolic Model Checker. Int. J. Softw. Tools Technol. Transf. 2(4), 410–425 (2000), https://doi.org/10.1007/s100090050046
10. Clarke, E.M., Emerson, E.A.: Design and Synthesis of Synchronization Skeletons Using Branching Time Temporal Logic. In: Grumberg, O., Veith, H. (eds.) 25 Years of Model Checking - History, Achievements, Perspectives. Lecture Notes in Computer Science, vol. 5000, pp. 196–215. Springer (2008)
11. Clearsy: https://www.atelierb.eu/en/presentation-of-the-b-method/formal-proof-presentation/
12. Consortium, E.B.: http://www.event-b.org/
13. Cunha, A., Macedo, N., Kang, E.: Task Model Design and Analysis with Alloy. In: Glässer, U., Campos, J.C., Méry, D., Palanque, P.A. (eds.) Rigorous State-Based Methods - 9th International Conference, ABZ 2023, Nancy, France, May 30 - June 2, 2023, Proceedings. Lecture Notes in Computer Science, vol. 14010, pp. 303–320. Springer (2023)
14. Event-B Consortium: SMT Solvers Plug-in. https://wiki.event-b.org/index.php/SMT_Solvers_Plug-in
15. Geleßus, D., Stock, S., Vu, F., Leuschel, M., Mashkoor, A.: Modeling and Analysis of a Safety-Critical Interactive System Through Validation Obligations. In: Glässer, U., Campos, J.C., Méry, D., Palanque, P.A. (eds.) Rigorous State-Based Methods - 9th International Conference, ABZ 2023, Nancy, France, May 30 - June 2, 2023, Proceedings. Lecture Notes in Computer Science, vol. 14010, pp. 284–302. Springer (2023)
16. Lamport, L.: Real-Time Model Checking Is Really Simple. In: Proceedings CHARME 2005. pp. 162–175 (2005)
17. Leuschel, M., Bendisposto, J., Dobrikov, I., Krings, S., Plagge, D.: From Animation to Data Validation: The ProB Constraint Solver 10 Years On. In: Boulanger, J.L. (ed.) Formal Methods Applied to Complex Systems: Implementation of the B Method, chap. 14, pp. 427–446. Wiley ISTE, Hoboken, NJ (2014)
18. Leuschel, M., Butler, M.J.: ProB: An Automated Analysis Toolset for the B Method. International Journal on Software Tools for Technology Transfer 10(2), 185–203 (2008)
19. Mammar, A., Frappier, M.: Modeling of a Speed Control System using Event-B. In: Raschke, A., Méry, D., Houdek, F. (eds.) Rigorous State-Based Methods - 7th International Conference, ABZ 2020, Ulm, Germany, May 27-29, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12071, pp. 367–381. Springer (2020)
20. Mammar, A., Frappier, M., Fotso, S.J.T., Laleau, R.: A Formal Refinement-Based Analysis of the Hybrid ERTMS/ETCS Level 3 Standard. Int. J. Softw. Tools Technol. Transf. 22(3), 333–347 (2020)
21. Mammar, A., Frappier, M., Laleau, R.: An Event-B Model of an Automotive Adaptive Exterior Light System. In: Raschke, A., Méry, D., Houdek, F. (eds.) Rigorous State-Based Methods - 7th International Conference, ABZ 2020, Ulm, Germany, May 27-29, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12071, pp. 351–366. Springer (2020)
22. Mammar, A., Leuschel, M.: Modeling and Verifying an Arrival Manager using Event-B. In: Glässer, U., Campos, J.C., Méry, D., Palanque, P.A. (eds.) Rigorous State-Based Methods - 9th International Conference, ABZ 2023, Nancy, France, May 30 - June 2, 2023, Proceedings. Lecture Notes in Computer Science, vol. 14010, pp. 321–339. Springer (2023)
23. Mammar, A., Leuschel, M.: Modeling and Verifying an Arrival Manager using Event-B. Available at https://github.com/AmelMammar/AMAN (January 2024)
24. Mammar, A., Laleau, R.: Modeling a Landing Gear System in Event-B. Int. J. Softw. Tools Technol. Transf. 19(2), 167–186 (2017)
25. Palanque, P.A., Campos, J.C.: AMAN Case Study. In: Glässer, U., Campos, J.C., Méry, D., Palanque, P.A. (eds.) Rigorous State-Based Methods - 9th International Conference, ABZ 2023, Nancy, France, May 30 - June 2, 2023, Proceedings. Lecture Notes in Computer Science, vol. 14010, pp. 265–283. Springer (2023)
26. Parnas, D.L., Madey, J.: Functional Documents for Computer Systems. Science of Computer Programming 25(1), 41–61 (1995)
27. Pnueli, A.: The Temporal Logic of Programs. In: 18th Annual Symposium on Foundations of Computer Science, Providence. pp. 46–57. IEEE Computer Society (1977)
28. Rehm, J., Cansell, D.: Proved Development of the Real-Time Properties of the IEEE 1394 Root Contention Protocol with the Event B Method. In: ISoLA. pp. 179–190 (2007)
29. Vu, F., Happe, C., Leuschel, M.: Generating interactive documents for domain-specific validation of formal models. Int. J. Softw. Tools Technol. Transf. 26(2), 147–168 (2024), https://doi.org/10.1007/s10009-024-00739-0
30. Werth, M., Leuschel, M.: VisB: A Lightweight Tool to Visualize Formal Models with SVG Graphics. In: Raschke, A., Méry, D., Houdek, F. (eds.) Proceedings ABZ 2020. pp. 260–265. LNCS 12071 (2020)

A An excerpt of the requirements

Requirement label	description
Req1	Planes can added to the flight sequence e.g. planes arriving in a closerange of the airport.
Req2	Planes can be removed from the flight sequence e.g. planes changing their landing airport for some reason.
Req3	Planes moved earlier or later on the timeline by the PLAN ATCo thus requiring from AMAN the processing of a new prediction.
Req4	Planes put on hold by the PLAN ATCo. Planes removed from HOLD will appear as normal aircrafts handled by AMAN.
Req5	Aircraft labels should not overlap.
Req6	An aircraft label cannot be moved into a blocked time period.
Req7	Moving an aircraft label might not be accepted by AMAN if it would require a speed up of the aircraft beyond the capacity of the aircraft.
Req8	If AMAN is not functioning (e.g. no update after 10 seconds) the ATCo must be informed about the failure and landing sequence preparation will be done manually (without AMAN).
Req9	Displays should provide recognisable feedback on entries made by the ATCo into the system so that the ATCo can detect and correct errors.
Req10	Displays should provide recognisable feedback on present state of the automated system or mode of operation. (What is it doing?).
Req11	Displays should provide recognisable feedback on actions taken by the system to achieve or maintain a desired state. (What is it trying to do?).
Req12	Displays should provide recognisable feedback on future states scheduled by the automation. (What is it going to do next?).
Req13	Displays should provide recognisable feedback on transitions between system states.
Req14	The set of tasks identified must be feasible on the interactive systems;this may be ensured by checking behavioural equivalence of the task model with respect to a model of the interactive application.
Req15	The HOLD button must be available only when one aircraft label is selected.
Req16	The zoom value cannot be bigger than 45 and smaller than 15.
Req17	Aircraft labels must always be positioned in front of a small bar of the timeline.
Req18	Lift of the zoom slider should always be located on the slider bar.
Req19	The value displayed next to the zoom slider must belong to the list of seven acceptable values for the zoom.
Req20	Each movement of the mouse on the ATCo table must be reflected by a movement of the cursor on the screen
Req21	There must be one and only one mouse cursor on the screen
Req22	Hold(aircraft) function can only be triggered after a mouse press and a mouse released have been performed on the HOLD button.
Req23	Hold(aircraft) function must not be triggered if there is not a mouse press and a mouse released performed on the HOLD button.

Table 3: An excerpt of the requirements from [25]

B B symbols

Table 4 gives the semantics of the different mathematical symbols used in the paper where:

- A and B denote any sets of elements,
- If a and b are elements of A and B respectively, $a \mapsto b$ denotes the tuple (a, b),

– A_1 and B_1 denote any subsets of A and B respectively,
– P denotes a predicate,
– S denotes any set expression.

Concept	Notation	Semantics
$A_1, \ldots, A_n$ is a partition of A	**partition** $(A, A_1, \ldots, A_n)$	$A_i \subseteq A \wedge \bigcup_i A_i = A$ $\forall i, j.i \neq j \Rightarrow A_i \cap A_j = \varnothing$
Set of parties of A	$\mathbb{P}(A)$	$\mathbb{P}(A) = \{A_1 \cdot A_1 \subseteq A\}$
Set of non empty parties of A	$\mathbb{P}1(A)$	$\mathbb{P}1(A) = \{A_1 \cdot A_1 \subseteq A \wedge A_1 \neq \varnothing\}$
Quantified union	$\bigcup z.\ P \mid S$	if $\forall z.(P \Rightarrow S \subseteq T$, then $\bigcup z.\ P \mid S = \{\ y \mid y \in T \wedge \exists\ z.(\ z \in T \wedge P \wedge y \in S)\}$
Lambda expression	$\lambda\ x.(x \in T \wedge P \mid E)$	If $\forall\ x.(x \in T \Rightarrow E \in U)$, $\lambda\ x.(x \in T \wedge P \mid E) = \{\ x, y \mid x, y \in T \times U \wedge P \wedge y = E\}$
R is a relation from A to B	$R \in A \leftrightarrow B$	$R \subseteq \{a \mapsto b \cdot a \in A \wedge b \in B\}$
R^{-1} is the inverse of R	R^{-1}	$R^{-1} = \{b \mapsto a \cdot a \mapsto b \in R\}$
Difference	$R_1 \backslash R_2$	if $R_1 \in A \leftrightarrow B$ and $R_2 \in A \leftrightarrow B$ then, $R_1 \backslash R_2 = \{a \mapsto b \cdot a \mapsto b \in R_1 \wedge a \mapsto b \notin R_2\}$
Override of R_1 by R_2	$R_1 \leftarrowtail R_2$	if $R_1 \in A \leftrightarrow B$ and $R_2 \in A \leftrightarrow B$ then, $R_1 \leftarrowtail R_2 = \{a \mapsto b \cdot a \mapsto b \in R_2 \vee (a \mapsto b \in R_1 \wedge a \notin dom(R_2))\}$
Direct product of R_1 and R_2	$R_1 \otimes R_2$	if $R_1 \in A \leftrightarrow B$ and $R_2 \in A \leftrightarrow C$ then, $R_1 \otimes R_2 = \{a \mapsto (b \mapsto c) \cdot a \mapsto b \in R_1 \wedge a \mapsto c \in R_2\}$
Image of A_1 by R	$R[A_1]$	$R[A_1] = \{b_1 \cdot (b_1 \in B \wedge \exists a_1 \cdot (a_1 \in A_1 \wedge a_1 \mapsto b_1 \in R))\}$
Domain of R	$dom(R)$	$dom(R) = \{a_1 \cdot (a_1 \in A \wedge \exists b_1 \cdot (b_1 \in B \wedge a_1 \mapsto b_1 \in R))\}$
Range of R	$ran(R)$	$ran(R) = \{b_1 \cdot (b_1 \in B \wedge \exists a_1 \cdot (a_1 \in A \wedge a_1 \mapsto b_1 \in R))\}$
Domain restriction of R	$A_1 \lhd R$	$A_1 \lhd R = \{a \mapsto b \cdot (a \mapsto b \in R \wedge a \in A_1)\}$
Partial function f	$f \in A \nrightarrow B$	$f \in A \leftrightarrow B \wedge \forall a \cdot (a \in A \Rightarrow card(f[\{a\}]) \leq 1)$
Total function f	$f \in A \rightarrow B$	$f \in A \nrightarrow B \wedge dom(f) = A$
skip substitution	**skip**	Do nothing: all the variables are not modified.
ANY substitution	**ANY** X **WHERE** G **THEN** S **END**	Execute the substitution S for any values of parameters X that satisfy G
Guard of an event E	$grd(E)$	If E=**ANY** X **WHERE** G **THEN** S then, $grd(E) = G$

Table 4: Some Event-B symbols and their semantics