Integration of iUML-B and UPPAAL Timed Automata for Development of Real-Time Systems with Concurrent Processes

Fatima Shokri-Manninen[1(✉)], Leonidas Tsiopoulos[1,2], Jüri Vain[2(✉)], and Marina Waldén[1]

[1] Åbo Akademi University, Turku, Finland
{fatemeh.shokri,marina.walden}@abo.fi
[2] Tallinn University of Technology, Tallinn, Estonia
{leonidas.tsiopoulos,juri.vain}@taltech.ee

Abstract. Developing safety-critical systems requires to consider safety and real-time requirements in addition to functional requirements. Event-B is a formalism that is visualised by iUML-B and supports the development of functional aspects having rich verification and validation tools. However, it lacks well-established support for timing analysis. UPPAAL Timed Automata (UTA), on the other hand, address timing aspects of systems, and enable model checking reachability and timing properties. By integrating iUML-B and UTA, we combine the best verifying and validating practices from the two methods achieving a formal development of systems. We present the mapping for translating iUML-B constructs to UTA. The novel aspect is the use of a multi-process trigger-response pattern to address the modelling and verification of reachability properties of complex systems with concurrent processes. The approach is demonstrated on an airport control system, where timing, fairness, as well as liveness properties play a vital role in proving safety requirements.

Keywords: Verification · Model checking · Timed automata · Event-B · iUML-B · UPPAAL · Real-time systems · Trigger-response patterns

1 Introduction

Correct-by-Construction Design (CCD) [1] plays an important role in the development of safety critical-systems, since it guarantees their reliability and correctness with respect to the system requirements. This is vital in cases where human safety and large financial assets are at stake. Correctness by construction is gained by the use of formal methods, which are mathematical methods for deriving a system based on its requirements. The main reasons for applying formal modelling is to avoid ambiguity or misunderstanding of system requirements and to detect problems early in system development.

A. Raschke et al. (Eds.): ABZ 2020, LNCS 12071, pp. 186–202, 2020.
https://doi.org/10.1007/978-3-030-48077-6_13

One of the formal methods which supports CCD in the development process is Event-B [1]. Event-B is based on set theory and supports design by stepwise refinement. Event-B tool RODIN with its plug-ins provides a rich support for this CCD. However, in spite of these beneficial aspects, Event-B lacks sufficient support for timing analysis and refinement of timed specifications. UTA [3] address timing aspects of systems providing efficient data structures and algorithms for their representation and verification, but are less focusing on supporting the refinement-based development and verification.

The goal of this paper is to advocate a model-based design method, where Event-B and UTA are combined to mutually complement each other. The motivation for selecting Event-B as the base formalism is that Event-B provides support for verifying infinite-sized models with advanced data structures using first-order logic. Additionally, it allows for correct-by-construction development via stepwise refinement. For mapping, we opt for UTA as it supports verification of real-time properties which are required before the implementation phase of the model. The design method consists of stepwise refining the system using Event-B for proving the functional and safety properties in each step. Each development step is then translated to a UTA model that can be validated via model checking and checked for real-time properties without re-checking non-timing related properties.

For the work in this paper we extend the earlier work on the Event-B to UTA mapping by introducing an intermediate representation in iUML-B for the generation of the control structure that serves as skeleton also for UTA. In particular, we investigate how the integrated approach addresses the development of complex real-time trigger-response pattern-based systems with concurrent processes and discuss the benefits of the integrated method in contrast to that when only one of the two formalisms is applied for the development and verification of safety critical Cyber-Physical Systems (CPS) [12,13].

2 Related Work

Formal development of CPSs requires continuous timing properties to capture real behaviour of these systems. As discrete timing constraints cannot describe some of the substantial dynamic properties like Zeno behaviour, essential discontinuities, and other singularities of real-time systems in the physical environment, continuous time constraints become the inseparable part of modelling CPS.

Event-B is a formal method for modelling a safety critical-system that originally lacks the notion of real-time. Recent attempts [5,8,10] have been made to integrate discrete time to Event-B using patterns, such as *delay*, *expriry*, *deadline* and *interval*. Since invariants on discrete time introduce noise to the provers [5], it easily leads to cases that are difficult to prove. All these timing properties contain a trigger and response pattern, which are modelled as events in Event-B. To capture all these timing properties we focus on their underlying trigger-response pattern, where a trigger must always be followed by a response.

Due to the lack of concept of time continuity, the above discrete time properties cannot always be applied to CPSs to embrace their continuous behaviour.

Zhu et al. addressed this problem by extension of deadline constraints [12]. Moreover, authors defined discrete task- and scheduler-based timing properties of each process and of concurrent tasks between processes, respectively. They refined task-based timing into scheduler-based timing by either a FIFO queue scheduling policy or a deferrable priority-based scheduling policy with aging. For addressing intermediate events between trigger and response events, they propose in [13] the conditional convergent notion. In this approach, intermediate events can converge if there is no response event enabled, assuming weak-fairness of intermediate events and eventual execution of the response event. While [13] addresses a single-process trigger-response pattern, we can model multiple trigger-response relations in UTA by applying our integrated approach. It allows verifying that the interleavings between concurrent processes do not cause deadlock while proving reachability, liveness and non-Zenoness properties in the model.

Compared to earlier research on combining Event-B and UTA [4,11], we extend the mapping by considering sequencing of Event-B events in iUML-B diagrams, and control structures representing trigger-response patterns. Specifically, the iUML-B graphical design provides us with an untimed control structure identical to that of UTA. This is further elaborated by incorporating timing analysis at each refinement/design step. The straight-forward mapping proposed in [4] leads to too large models. In a later approach [11] an event-level mapping was introduced where each event from the Event-B specification was translated to an UTA and then parallely composed to form the full model. Additional optimisations were needed to aggregate the automata which model mutually exclusive events, and thus, reduce interleaving of model events.

In this paper the mapping is still based on the events but the UTA model structure is extracted from the iUML-B state diagram. By decorating the extracted control structure with UTA specific attributes we can verify the system's timing correctness and provide feedback to the Event-B side of the development for the feasibility of system events. For capturing the behaviour of processes based on the trigger-response pattern, we show in the following sections how concurrency with multi-process intermediate events is modeled and verified using an airport control system as a case study.

3 Preliminaries

3.1 Event-B and iUML-B

Event-B [1] is a state-based formalism for the development of reactive and distributed systems. Event-B uses refinement [2], which enables the system to be created in a stepwise manner gradually adding details into the model proving that each refinement step preserves the correctness of the previous steps. A model in Event-B, a machine, can be interpreted as a transition system where the variable valuations constitute the states and the events represent the transitions. Machines can be refined either via *superposition refinement* [9], where new features are added to the machine, or by *data refinement*, where abstract

features are replaced by more concrete ones. Event-B is well supported by the Rodin Platform [6], which is extendable with plugins facilitating the modelling and verification.

iUML-B is an integrated form of the classical UML-B graphical front-end for Event-B [7] that is an extension of the Rodin Platform. It allows modellers to build a model through a diagrammatic design in the form of state-machines and class diagrams. The translator then generates Event-B automatically facilitating the modelling process. Class diagrams provide a way to model data relationships, while state-machines show the states and transitions of an Event-B machine. The guards and actions of the Event-B events form the guards and actions of the transitions in the state-machine diagrams. The operational semantics of the events are, hence, visualised with the state-machines.

In a state-machine with an transition e_1 between states S_1 and S_2, transition e_1 can be fired if the state is S_1 and the guard of the transition $G(t, v)$ evaluates to *true*. When e_1 is fired it changes the state to S_2 and may also modify other variables of the state-machine via actions $S(t, v)$. This corresponds to event e_1 in Event-B:

$$e_1 = \textbf{any}\ t\ \textbf{where}\ state = S_1 \wedge G(t, v)\ \textbf{then}\ state := S_2 \mid\mid S(t, v)\ \textbf{end}$$

Invariants may also be given in the states. They correspond to invariants in an Event-B machine. The state-machines can be refined in a corresponding manner to the Event-B machines concerning variables and events. Additionally, states can be nested in state-machines (i.e. states in a state), which is also often used when refining a system to model the increased level of detail in the states.

3.2 UPPAAL Timed Automata

UTA [3] are defined as a closed network of extended timed automata that are called processes. The processes are combined into a single system by synchronous parallel composition like that in process algebra CCS. The nodes of the automata graph are called locations and directed lines between locations are called edges. For each edge, which is a transition between two locations, conditions or guards can be defined. Whenever the guard holds, the edge can be fired, which leads to a new location. Communication and synchronisation between different automata is taken care of by *send* and *receive* actions. An action *send* over a channel h is denoted by $h!$ and its co-action, *receive* is denoted by $h?$.

Formally, an UTA is defined as the tuple $(L, E, V, CL, Init, Inv, T_L)$, where:

- L is a finite set of locations,
- E is the set of edges defined by $E \subseteq L \times G(CL, V) \times Sync \times Act \times L$, where
  - $G(CL, V)$ is the set of constraints in guards,
  - $Sync$ is a set of synchronisation actions over channels and
  - Act is a set of sequences of assignment actions with integer and boolean expressions as well as with clock resets.

- V denotes the set of integer and boolean variables,
- CL denotes the set of real-valued clocks ($CL \cap V = \varnothing$),
- $Init \subseteq Act$ is a set of assignments that assigns the initial values to variables and clocks,
- $Inv : L \rightarrow I(CL, V)$ is a function that assigns an invariant to each location, $I(CL, V)$ being the set of invariants over clocks CL and variables V and
- $T_L : L \rightarrow \{ordinary, urgent, committed\}$ is the function that assigns the type to each location of the automaton.

In *urgent* locations an outgoing edge will be executed immediately when its guard holds. *Committed* locations are useful for creating atomic sequences of process actions since an outgoing edge must be executed immediatelly without time passing.

UTA Requirement Specification Language. The requirement specification language (in short, query language) of UTA, used to specify properties to be model checked, is a subset of Timed Computation Tree Logic (TCTL) [3]. The query language consists of path formulae and state formulae. State formulae describe individual states, whereas path formulae quantify over paths or traces of the model and can be classified into *reachability*, *safety* and *liveness* [3]. For example, safety properties are specified with path formula $A\Box\varphi$ stating that state formula φ should be true in all reachable states. In the next section we describe in more detail the TCTL formulae we apply in the rest of this paper.

4 Mapping from Event-B and iUML-B Models to UTA

We base our work here on the previous work by Vain et al. [11]. We assume that the system is developed stepwise using Event-B and iUML-B and prove the safety properties in each step using the proof system of this formalism. The result of each development step is then translated to UTA in order to have a model that can be validated via model checking and specifically checked for real-time properties avoiding re-checking of functional/safety properties. Note that due to the locality of refinements only those model fragments that are introduced by Event-B refinements need to be mapped to the corresponding UTA fragments. The rest of the UTA model defined in earlier steps remains untouched by the current Event-B refinement step.

Plant and Controller. In Event-B and iUML-B the model represents a holistic view to the control systems where the controller and the plant events are all given in one machine. However, when mapping the model to UTA these different kinds of events have to be identified. The *plant events* in the iUML-B state-machines are mapped to UTA plant automaton with corresponding states and transitions. This leads to a sequential model of the control system in UPPAAL. The states and state transitions of the state-machines are given in Event-B as global, but when mapped to UTA the states and transitions are partitioned by automata that introduces modularity to models and to verification. The *controller events* are each translated as in [11] to simple self-loop automata. All these events

emulating self-loop transitions are composed in parallel. The communication with the plant takes place via channels by trigger and response actions.

In control systems, there might be several plants (processes) for a controller. In Event-B the setup of the system is given in the context machine. Only one state-machine is created for the system, but the plants/processes are specified as instances of the machine. When mapping this scenario to UPPAAL, one UTA template is created for the process and instantiated for the multiple processes.

Mapping of Functions and Predicates. Variables of integer and enumerated types in Event-B become integers in UTA, while finite sets and relations in Event-B are mapped to (multidimensional) arrays in UTA. We can then implement the set and relational operators as C-functions in UTA.

Mapping of Events. Transitions in iUML-B are generally translated to state transitions in UTA [11]. In Fig. 1 we exemplify the translation with an iUML-B state machine and Event-B code to the left and a corresponding UTA model to the right. Let

$$e = \textbf{any}\ p\ \textbf{where}\ G(p, v)\ \textbf{then}\ S(p, v)\ \textbf{end}$$

be an event of Event-B, then

(i) the parameter p will appear in the select label of the UTA edge, which contains a comma separated list of **p : int** expressions where p is a variable name and *int* is a defined type (see Fig. 1).
(ii) the event guard *G(p,v)* is mapped to the guard *G(V)* of an edge where V denotes UTA variables corresponding to variables v (**p > 5** in Fig. 1).
(iii) the event action *S(p,v)* corresponds to assignment statements (updates) *V' = S(V)* of the UTA edge (**num := num + p** in Fig. 1).

For plants consisting of many processes, the instance of a plant is identified by a unique parameter value. The template may have parameters of type integer. This allows modelling the ANY-construct of Event-B, where the choice is *finite*. The parameter of a template specified by its type defines the instances (processes) of the template, one for each value in the parameter type.

Timing of Events. When mapping the Event-B model to UTA, we need to add timing explicitly to the model to be able to consider timing aspects like time-bounded reachability. When adding explicit timing constraints to UTA events, it is assumed that the occurrence of an event is instantaneous. An event may occur within some time interval $[lb, ub]$, where lb stands for lower bound and ub stands for upper bound, provided it is enabled by guard G. For specifying these constraints, new variables, namely the set CL of clocks is introduced. We usually assume the continuous intervals are of shape $[lb, ub]$, where lb and $ub \geq 0$, and $ub \geq lb$. Note that having infimum inf of clocks domains $inf\ dom(cl) = 0$ and guards $cl \geq inf$ it may introduce Zeno computations if there exists a loop in the model where the maximum of lower bounds of occurrence interval of each edge is equal to the infimum.

In general, the timing specification of events introduces bounded intervals of occurence that are specified as location invariant $\mathsf{inv}(CL) \equiv \wedge_i cl_i \leq ub_i$ and the guard $G_i(CL) \equiv \wedge_j cl_j \geq lb_j$ of its self-loop edge e_i that models an event. A set of clock conditions $G_i(CL)$ and $\mathsf{inv}(CL)$ indicate time constraints when an event e_i should be fired, i.e. not later than time ub_i (deadline) and not before time lb_i (delay) (see $cl <= ub$ and $cl >= lb$ in the UTA in Fig. 1).

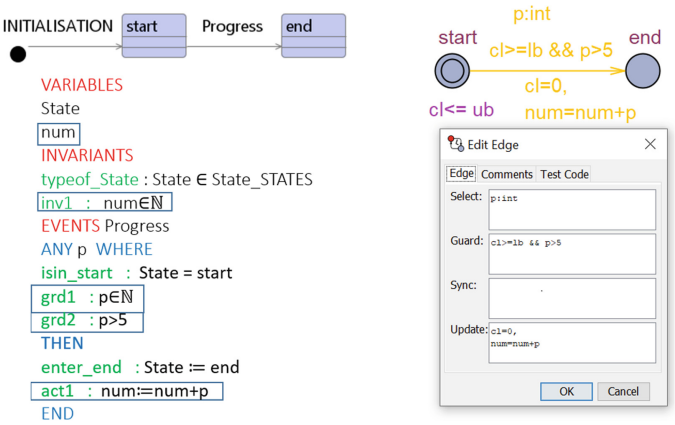

Fig. 1. Transition structure in iUML-B (left) and UTA (right)

Undelayed Reaction. In the context of multiple trigger-response patterns, some response should be fired immediately without delay in a critical situation. It brings the concept of priority based on timing. According to the timing constraint proposed in [8], the delay constraint is specified by $Delay(Trigger, Response, delay)$ that specifies the delay constraint where $delay = 0$. In UPPAAL, undelayed response is modelled with *urgent channel* (urgent chan) which is defined to be synchronizing executions of enabled edges without delay. Clock conditions on these transitions are not allowed. An alternative to model undelayed reaction encoded as a single (not synchronized) edge is to define its source location type as either *committed* or *urgent* or set this location invariant condition upper bound to 0.

Invariants. The invariants of Event-B are not directly translated into UPPAAL model-checking queries. However, these invariants can be specified and model checked as TCTL formulas of form $A\Box p$, where p is a first order state formula and the pair of modalities $A\Box$ requires p to be true in all reachable states of the model in UPPAAL. Formula p can involve predicates on model clocks to specify explicit timing constraints.

Time Bounded Reachability. For real-time applications, we consider time bounded reachability as one of the most fundamental properties. In UTA, the reachability of an event E (where E is specified in terms of after state of the

event and/or valuation of state variables) from model initial state is expressible using TCTL formula pattern A$\Diamond E$ && $Clock \leq$ TB, for time bound TB.

Trigger-response properties are expressed as a special case of time bounded reachability where the reachability of a response event is always considered relative to its trigger events. Proving multiple trigger-response properties in Event-B presumes augmentation of the model with auxiliary "boolean property variables" which are set to true only when considered triggers receive a response (as $Landing\_permission(selfP) := TRUE$ in Fig. 5). Conjoining multiple trigger-response pairs of different processes is non-trivial and can easily cause misinterpretation. For instance, reactions to stimuli of different processes may occur in different states or even be mutually exclusive in some states. Therefore, the trigger-response properties for multiple concurrent processes need to be specified and checked separately.

The reachability of a response event (actually its post condition rp) from a trigger event tr (respectively its post condition tr) is then expressed in UTA and TCTL using *leads to* operator as $tr \dashrightarrow rp$. The multiple trigger-response properties can be specified and proved similarly. For instance, TCTL formula $tr_1\&\&...\&\&tr_n \dashrightarrow rp_1\&\&...rp_m$ where auxilliary boolean variables $tr_1, ..., tr_n$ in the model register the occurrence of trigger 1 to trigger n and auxilliary variables $rp_1, ..., rp_m$ register the occurrence of response events 1 to m.

In case of time bounded reachability of a response event, a property clock constraint should be conjoined to the right hand side of *leads to*. Here it should be granted that the property clock is reset in the model at the moment when the trigger (conjunction $tr_1\&\&...\&\&tr_n$) of considered trigger-response pair is set to true.

Liveness. In Event-B due to weak fairness, enabled processes will eventually be executed. If the system is deadlock free, there is always an event that can be executed. In UTA, the situation, where a transition is enabled but there is no finite interval specified in the location invariant (or not using location types *urgent*, *committed*), may result in an infinite waiting in that location. This provides behaviourally similar effect as deadlock. It means that regardless if one or more of the outgoing edges of that location are enabled, none of them will ever be executed because there is no upper bound that forces the edge to be executed in finite time. In that way, weak fairness is not sufficient to guarantee non-blocking in UTA and the progress must be granted by specifying the location type either *committed* or *urgent* or adding a time bound conjunct to each location invariant. Liveness can be proved by TCTL query A$\Box$ *not deadlock* provided all legal terminal locations are supplied with self-loop edges.

5 Overview of Case Study

In order to demonstrate the CCD methodology with integrated formal methods, we use an airport control system example. We propose this case study for presenting the verification of behavioral and timing properties by combining two complementing formalisms Event-B and UTA. We focus only on the landing

control with one runway. Based on system requirements, we have two types of landing, namely, emergency landing and normal landing. The flight control is in charge of safe landing by giving airplanes permission to land at appropriate times. For normal landing planes may queue up to enter the landing runway. There are two queues with different priorities based on the planes' fuel level. An emergency landing has higher priority than both queues. In case of an emergency landing, no other plane can land and all landing requests are rejected. Only one emergency landing can take place at a time. As a safety requirement it is ensured that there is no plane in the runway before allowing another plane to use it.

The model of the airport system[1] consists of one abstract model with three refinement steps. The abstract model presents the general view of the airport system, with two landing modes. In the first refinement step, we introduce two queues with different priorities for normal landing. Next, we implement the FIFO policy of the queues. Finally, we introduce fuel level for each plane.

5.1 M0: An Abstract Model of Airport Control System

In the abstract model the behaviour of the system is depicted in the a state-machine diagram of iUML-B (in Fig. 2) where the different states of a plane to reach the final state (*At_Gate*) are modeled. In Event-B, we create a context that introduces the set *PLANES* in addition to the generated implicit context which consists of the states in the state-machine. The state of a plane and the transitions between the states are generated automatically from the state-machine. We define a parameter $selfP$ in the iUML-B diagram to represent the instances of *PLANES*, which is translated into corresponding Event-B events.

In the abstract Airport Control System in Fig. 2, a plane in state In_Air sends a landing request to the controller. If the response is positive, the plane can enter the landing queue. We define an invariant ($LQ\_Permission = TRUE$) in the *Landing_Queue* state which ensures that each plane in the landing queue has a landing queue permission.

The controller checks whether there is an ongoing emergency case or not via the boolean variable *Emergency_Prog*. In case of an ongoing emergency, no new plane will get permission to land and will have to leave the airport.

We ensure safety on the critical section, the runway, using mutual exclusion with boolean variable *Runway_Busy* to ensure that no landing permission is admitted if there is a plane on the runway. When the plane is on the runway it will be given a gate by the controller and it moves to the final state *At_Gate*.

Emergency request can be sent in states *Landing_Queue* or In_Air. Only one emergency at a time can be handled at the airport. If there is already an emergency, emergency landing permission for that plane is rejected, and the plane has to leave the airport.

In order to be able to introduce timing properties to the airport system, we map the Event-B and iUML-B model into an UTA. In Event-B, the model of

[1] iUML-B and UTA models are found in: https://github.com/fshokri/FormalModels.git.

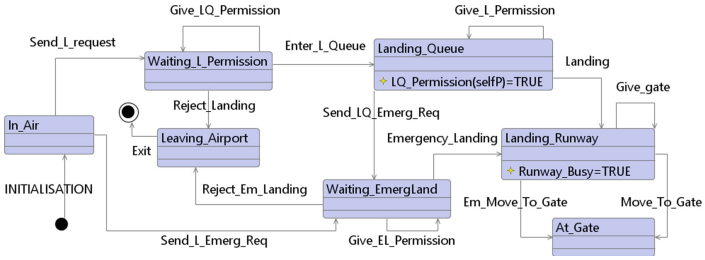

Fig. 2. The abstract model in iUML-B

the plane and the controller are integrated. However to model different timing behaviors of components in UPPAAL, we need to divide the model into plane and controller (Fig. 3) where simultaneous events are synchronised via channels.

For the plane template, we follow the same structure as in the iUML-B state-machine diagram, while for the controller we only consider events for giving permissions. The instantiation of the plane template for modelling the plane instances is modelled with template parameter *id*, while the handling of each plane by the controller is addressed with the *select* clause in UTA corresponding to the ANY event parameter in Event-B. This is described in Mapping of events in Sect. 4.

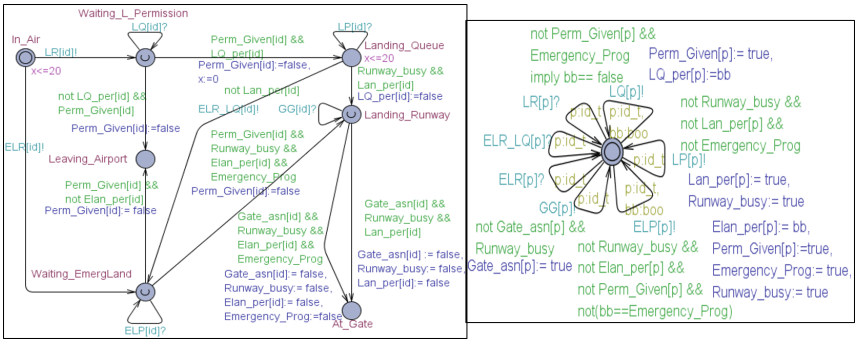

Fig. 3. The abstract model of plane (left) and controller (right) in UTA

We define clock constraints with upper bounds stated with invariants for locations waiting before triggering requests. We assign locations as *urgent* when waiting for responses from the controller. This way planes can progress immediately when a response is given. For the *Landing_Queue* location an upper bound is given allowing flexibility when later refining the behaviour for planes in the landing queues i.e., allowing time passing when queueing.

Comments on the Modelling. In Event-B, the controller in the abstract system gives landing queue entry permissions non-deterministically. Since there

is no implicit timing in Event-B, this does not create a deadlock. However, in order to avoid deadlock in the corresponding state *Waiting_L_Permission* of the UTA model, we define a variable (*Perm_Given*) to indicate that permission has been given to make exit conditions from this state more deterministic. In this way each plane gets an answer for the landing request, either a positive one to move to the landing queue or a negative one to leave the airport.

5.2 M1: Introducing Two Queues

In the first refinement, we split the state *Landing_Queue* into two queues states, *High_Priority_Queue* and *Low_Priority_Queue*. This is done by adding nested state-machines in iUML-B, while in UTA two separate states and transitions are created (Fig. 4).

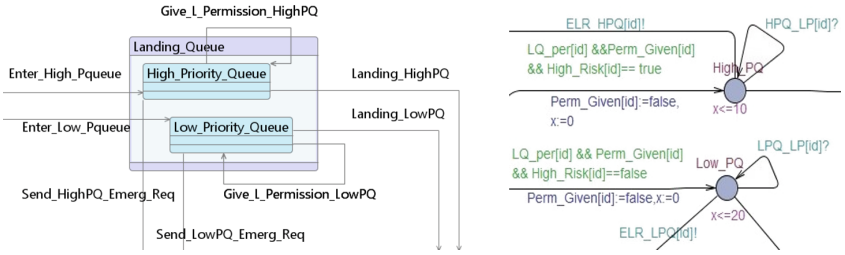

Fig. 4. The first level of refinement model excerpt in iUML-B (left) and UTA (right)

In M1, we introduce a new boolean variable *High_Risk* which states whether a plane has a high risk or not in the queueing situation. The plane with a high risk is eligible to move to *High_Priority_Queue* with shorter waiting time, while in normal situation planes enter *Low_Priority_Queue*. If *High_Priority_Queue* is not empty, a plane in *Low_Priority_Queue* needs to wait for a number of planes (here at most three) in *High_Priority_Queue* to land.

The nested state with queues in Event-B is translated to a separate state for each queue in UTA. The guards and actions of the events in Event-B are translated in a straightforward manner to guards and updates in the UTA model.

5.3 M2: Implementing FIFO Method for Each Queue

In the second refinement step, we implement the FIFO policy for each queue. Since the functionality of two queues is the same, we focus on the high priority queue in the Event-B model (Fig. 5).

The queues have positions (@inv1) and are of limited length (@inv4). Via event *Enter_High_Pqueue*($selfP$) plane $selfP$ can enter the high priority queue *High_Pqueue* provided that it is a high-risk plane that has been given landing permission and that the high priority queue is not full. The plane $selfP$ will be

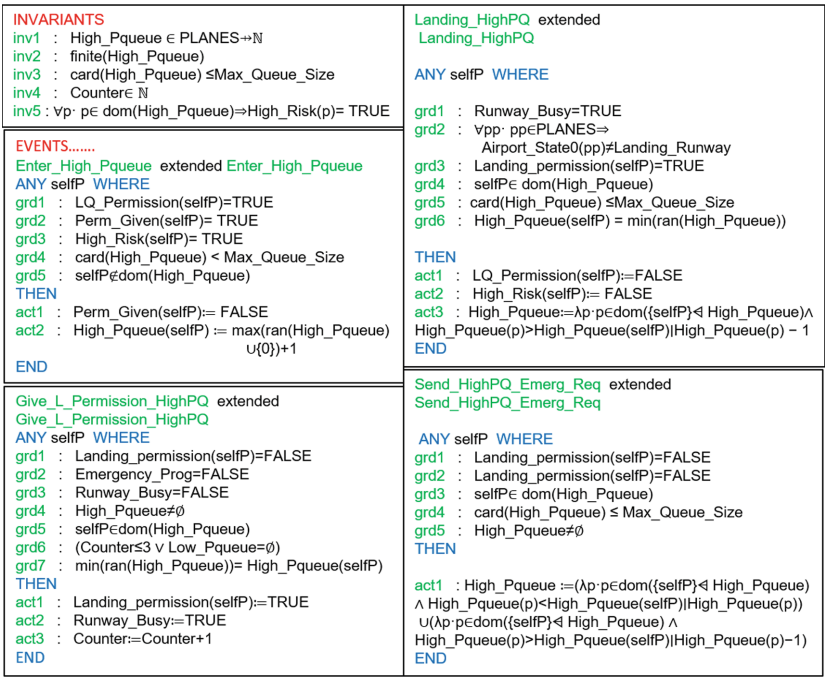

Fig. 5. The Event-B code for the second refinement with FIFO queue

inserted in the first free position of the queue which is position one if the queue is empty. In event $Give\_L\_Permission\_HighPQ$, landing permission is given to the first plane in the queue provided that less than three planes from the high priority queue have landed in a row or low priority queue is empty. Event $Landing\_High\_Pqueue(selfP)$ models plane $selfP$ leaving the high priority queue and entering the landing runway. As a result of landing, queueing planes are shifted in the queue. If there is an emergency situation while the plane is in the queue the plane will leave the queue (event $Send\_HighPQ\_Emerg\_Req$).

In the UTA model, we use functions for enqueuing and dequeuing, for a smooth implementation of the FIFO queue corresponding to the lambda expressions in Event-B. For example, action $act1$ of event $Send\_HighPQ\_Emerg\_Req$ in Fig. 5 is mapped to C-like functions in UTA as in Fig. 6. The left one (Em_deHPqu_idx) appears in the guard and the right one ($Em\_deHPqueue$) in the update of the transition from location $High\_PQ$ to $Waiting\_EmergLand$ in Fig. 8.

5.4 M3: Introducing Fuel Consumption

In the third refinement step, we introduce variables $Plane\_Fuel$ and $Fuel\_count$ in our Event-B model to indicate fuel consumption. Variable $Plane\_Fuel$ gives the fuel level (*High*, *Medium* and *Low*) for each plane, while $Fuel\_count$ is a

```
//Finding position of plane in queue
id_t Em_deHPqu_idx(id_t plane ) {
      int i=0;
    while (HPqueue[i] != plane)
 { i++;}
 return i;}
```

```
//Removing the plane based on its position
void Em_deHPqueue(id_t i ){
     --lenH;
     while (i < lenH) {
          HPqueue[i] = HPqueue[i + 1];
          i++;} HPqueue[i] = 0;}
```

Fig. 6. UTA C-like functions for the second refinement with FIFO queues

variant of type natural number to show that the superposed fuel consumption will not take over the behaviour of the system.

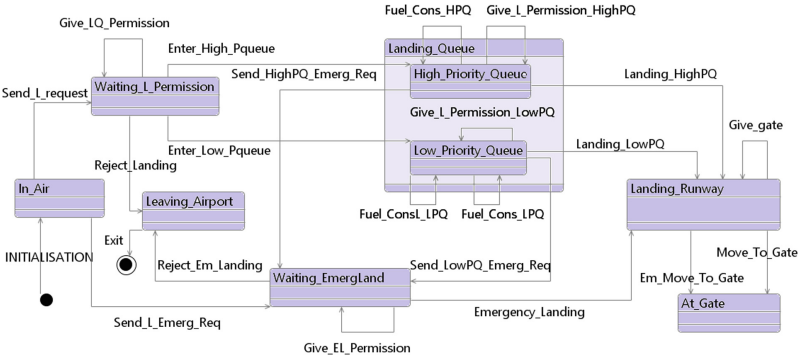

Fig. 7. The last level of refinement model in iUML-B

A plane with fuel level *Medium* enters *High_Pqueue*, while a plane with fuel level *High* is eligible for *Low_Pqueue*. Events *Fuel_Cons_HPQ*, *Fuel_Cons_LPQ* and *Fuel_ConsL_LPQ* will decrease plane fuel while waiting for permission to enter the landing runway (see Fig. 7). If the fuel level drops to *Low*, the plane will send an emergency request. The plane with the emergency request will reach the *At_Gate* location if there is no other emergency situation progressing.

The iUML-B state-machine is directly mapped to a UTA (see Fig. 8). However in the UTA model, the variable *Plane_Fuel* is mapped to a variable of type enumerated set which assigns a numerical value for the fuel level of each plane. The fuel consumption events are the events of the plane which are considered as intermediate events. The execution of these events depends on the delayed response from the controller for assigning landing permission. For avoiding delays on transitions triggering emergency cases, we defined the ELP, ELR_HPQ, ELR_LPQ channels in UTA to be *urgent*.

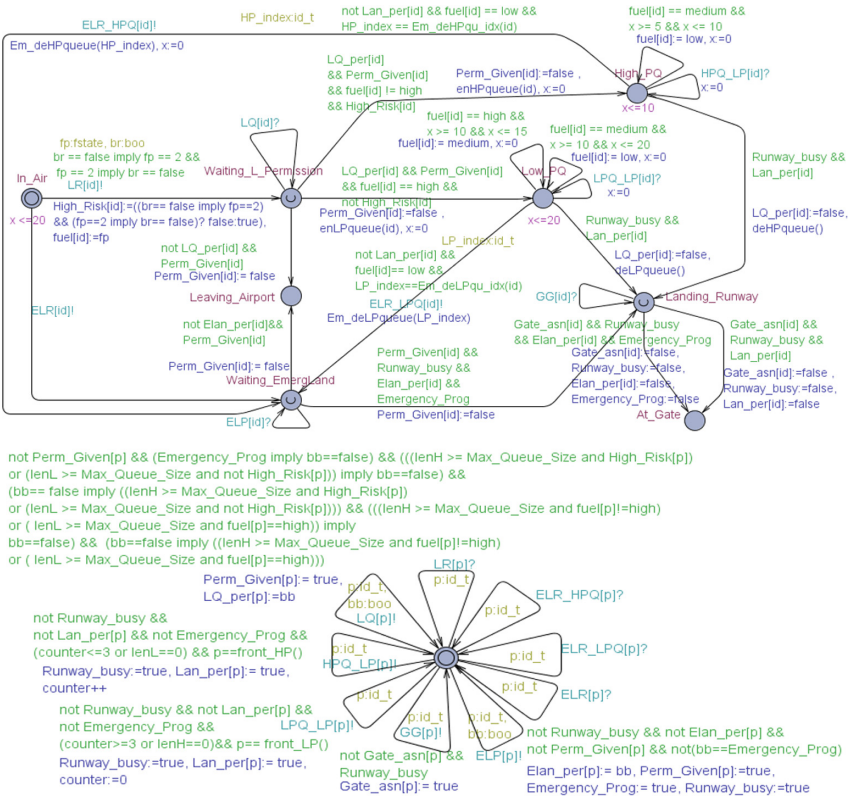

Fig. 8. The last level of refinement of Plane and Controller in UTA

5.5 Analysis Results

Proof Statistics: Machines M0 (abstract specification) and M1 (first refinement) in Event-B are automatically proved almost (100%) by the Rodin tools. For machine M2, where we introduce the FIFO mechanism for the queues, more complex proof obligations were generated of which 57% were automatically proved. In M3, where we define fuel consumption, 85% were automatically discharged. By triggering the interactive provers the rest of the proof obligations were discharged to get a fully proved model.

TCTL Queries: To verify the real-time behaviour of our UTA model based on multi-trigger and response pattern, we specify correctness properties in TCTL (Table 1). The properties which occur and need analysis in real systems include concurrency, deadlock freedom, non-Zenoness, liveness and reachability as well as the existence of intermediate events. Note that the mutual exclusion and fairness have already been proved using Event-B proof support. The timing related properties such as time bounded reachability and trigger-response timing

correctness are verified in UPPAAL. Some of the most characteristic timing properties of the case study are exemplified by Queries 1 to 3 in Table 1.

Table 1. TCTL queries based on multi-trigger and response pattern

Id	Query	Result
1	M3_Planes(1).Waiting_L_Permission && High_Risk[1] && fuel[1]! =high && not Emergency_Prog && lenH < Max_Queue_Size && LQ_per[1] --→ M3_Planes(1).High_PQ && aux_clk <= tb1	Satisfied
2	M3_Planes(1).Waiting_EmergLand && Elan_per[1] && Runway_busy && Gate_asn[1] --→ M3_Planes(1).At_Gate && aux_clk <= tb2	Satisfied
3	A<> forall (i : int [0,4]) Planes(i).At_Gate || M3_Planes(i).Leaving_Airport & & Gclk <= 120	Satisfied

Query 1 exemplifies a simpler timed trigger-response property satisfied by our system for plane instance 1. It states that when the plane will trigger its landing permission request to the controller, including also information about risk and fuel level, the response by the controller, if there is no ongoing emergency and if the queue is not full, will lead to the plane reaching location $High\_PQ$ within time $tb1$. Note that aux_clk is an auxiliary clock used only for the verification of the query. Constant $tb1$ comes from system requirements expressing the upper time bound explicitly for this trigger-response property.

Query 2 exemplifies a more complex timed trigger-response property, including intermediate events, satisfied by our system for plane instance 1. It states that when the plane triggers an emergency landing case to the controller entering location $Waiting\_EmergLand$ this will lead to the plane finally reaching location At_Gate if the intermediate response events by the controller allow it, i.e., the controller giving the emergency landing permission, keeping the runway reserved and assigning a gate to this plane instance. Application and assumptions for aux_clk and time bound $tb2$ are as for Query 1.

Query 3 represents the full integral time-bounded reachability property satisfied by our system. It states that all plane instances reach eventually the legal terminal locations At_Gate or $Leaving\_Airport$ by the time the global clock of the system reaches 120 time bound. The upper bound for the system global clock for this query is based on the real-time constraints on resolving the landing situation by traffic situations given by model constraints.

6 Conclusion and Discussion

The novel aspect studied in this paper is the use of multi-process trigger-response pattern with intermediate events to address the modelling and verification of

reachability properties of complex systems with concurrent processes. We first extended the Event-B to UTA mapping by incorporating iUML-B for the generation of the control structure that serves as the skeleton for UTA avoiding generation of too large UTA models. We then investigated how the integrated approach addresses the development of complex real-time trigger-response pattern-based systems with concurrent processes. We have shown that by using our integrated method we can address the development of complex real-time systems with concurrent processes without extending Event-B nor UTA standard features.

The co-use of Event-B and UTA and translation between them seems straight forward since we follow the control structure imposed by the iUML-B state-machine representation. Therefore, an automated translation from iUML-B is currently being investigated.

An essential observation is that introducing timing constraints by imposing them mechanically to Event-B or iUML-B model control structure often reveals modelling cases that are correct from untimed perspective, but may appear to be infeasible from the perspective of timing. For example, introducing non-zero durations to triggering conditions of events, may introduce some blocking conditions that results in violation of liveness properties proved on the initial Event-B model. Hence, the combination of the two approaches has proved to be beneficial to the development of coherent well-timed models.

Acknowledgments. This work has received funding from the Electronic Component Systems for European Leadership Joint Undertaking under grant agreement No 737494. This Joint Undertaking receives support from the European Union's Horizon 2020 research and innovation programme and Sweden, France, Spain, Italy, Finland, the Czech Republic. This was also supported by the ERDF funded centre of excellence project EXCITE (2014-2020.4.01.15-0018) and the Estonian Ministry of Education and Research institutional research grant no. IUT33-13.

References

1. Abrial, J.R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press, Cambridge (2010)
2. Back, R.J., von Wright, J.: Refinement Calculus: A Systematic Introduction. Graduate Texts in Computer Science. Springer, Heidelberg (1998). https://doi.org/10.1007/978-1-4612-1674-2
3. Behrmann, G., David, A., Larsen, K.G.: A tutorial on UPPAAL 4.0. Department of computer science, Aalborg university (2006)
4. Berthing, J., Boström, P., Sere, K., Tsiopoulos, L., Vain, J.: Refinement-based development of timed systems. In: Derrick, J., Gnesi, S., Latella, D., Treharne, H. (eds.) IFM 2012. LNCS, vol. 7321, pp. 69–83. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30729-4_6
5. Cansell, D., Méry, D., Rehm, J.: Time constraint patterns for event B development. In: Julliand, J., Kouchnarenko, O. (eds.) B 2007. LNCS, vol. 4355, pp. 140–154. Springer, Heidelberg (2006). https://doi.org/10.1007/11955757_13
6. Jastram, M., Butler, M.: Rodin User's Handbook: Covers Rodin v 2.8 (2014)
7. Said, M.Y., Butler, M., Snook, C.: A method of refinement in UML-B. Softw. Syst. Modeling **14**(4), 1557–1580 (2013). https://doi.org/10.1007/s10270-013-0391-z

8. Sarshogh, M.R., Butler, M.: Specification and refinement of discrete timing properties in Event-B. Electron. Commun. EASST **46**, 1–15 (2012)
9. Snook, C., Waldén, M.: Refinement of statemachines using Event B semantics. In: Julliand, J., Kouchnarenko, O. (eds.) B 2007. LNCS, vol. 4355, pp. 171–185. Springer, Heidelberg (2006). https://doi.org/10.1007/11955757_15
10. Sulskus, G., Poppleton, M., Rezazadeh, A.: An interval-based approach to modelling time in Event-B. In: Dastani, M., Sirjani, M. (eds.) FSEN 2015. LNCS, vol. 9392, pp. 292–307. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24644-4_20
11. Vain, J., Tsiopoulos, L., Boström, P.: Integrating refinement-based methods for developing timed systems. In: From Action Systems to Distributed Systems, pp. 199–214. Chapman and Hall/CRC (2016)
12. Zhu, C., Butler, M., Cirstea, C.: Refinement of timing constraints for concurrent tasks with scheduling. In: Butler, M., Raschke, A., Hoang, T.S., Reichl, K. (eds.) ABZ 2018. LNCS, vol. 10817, pp. 219–233. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-91271-4_15
13. Zhu, C., Butler, M., Cirstea, C.: Semantics of real-time trigger-response properties in Event-B. In: 2018 International Symposium on Theoretical Aspects of Software Engineering (TASE), pp. 150–155. IEEE (2018)