Refinement-Based Development of Timed Systems

Jesper Berthing[1], Pontus Boström[2], Kaisa Sere[2], Leonidas Tsiopoulos[2], and Jüri Vain[3]

[1] Danfoss Power Electronics A/S, Denmark
jbe@danfoss.com
[2] Åbo Akademi University, Finland
{pontus.bostrom,kaisa.sere,leonidas.tsiopoulos}@abo.fi
[3] Tallinn University of Technology, Estonia
vain@ioc.ee

Abstract. Refinement-based development supported by Event-B has been extensively used in the domain of embedded and distributed systems design. For these domains timing analysis is of great importance. However, in its present form, Event-B does not have a built-in notion of time. The theory of refinement of timed transition systems has been studied, but a refinement-based design flow of these systems is weakly supported by industrial strength tools. In this paper, we focus on the refinement relation in the class of Uppaal Timed Automata and show how this relation is interrelated with the data refinement relation in Event-B. Using this interrelation we present a way how the Event-B and Uppaal tools can complement each other in a refinement-based design flow. The approach is demonstrated on a fragment of an industrial case study.

1 Introduction

The Correct-by-Construction Design (CCD) workflow has proven its importance with motivating facts from recent industrial practice. Peugeot Automobiles has developed the model of the functioning of subsystems (lightings, airbags, engine, etc) for Peugeot after sales service; RATP (Paris Transportation) used the model of automatic platform doors to equip an existing metro line to verify the consistency of System Specification. Event-B [1] as one such CCD supporting formalism has proven its relevance in data intensive development while lacking sufficient support for timing analysis and refinement of timed specifications. Uppaal Timed Automata (UPTA) [2] address timing aspects of systems providing efficient data structures and algorithms for their representation and analysis but are less focusing on supporting refinement-based development, especially data refinement. The goal of this paper is to advocate the model-based transformational design method where these two approaches are combined to mutually complement each other.

The transformational design flow discussed in the paper consists of alternation between data and timing refinement steps (see Fig. 1). The result of a data

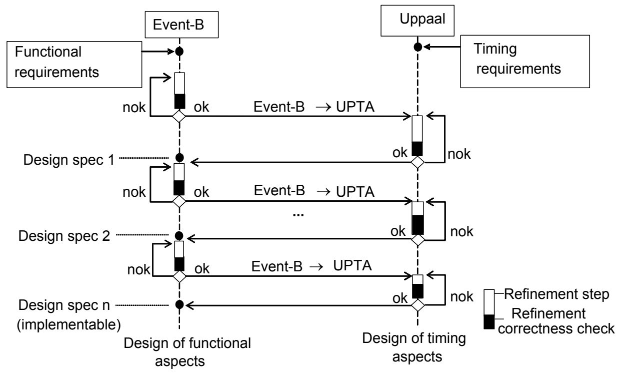


Fig. 1. CCD workflow with interleaving data and timing refinement steps

refinement step, performed within Event-B, after being proved correct, serves as an input to timing refinement step. The timing refinement means mapping the constraints of the previous level timing specification onto the UPTA model that is derived from the UPTA model of the previous design step and from the Event-B refinement of the current step. Then the newly introduced refinement of Event-B model must be decorated with timing attributes so that timing correctness criteria are satisfied. The timing correctness of refinement is verified using Uppaal [2] tool in two steps. At first, the consistency of the model being the result of timing refinement step is verified internally. Here the properties, e.g. deadlock and non-Zenoness are checked. Second, the preservation of timing properties introduced in the previous refinement step are verified. The design flow depicted in Fig. 1 is not complete since the real design flow may include also backtracking over several earlier design steps. For instance, when no feasible timing refinement is possible, it may require revision of much earlier functional refinement phases. The design backtracking issues and error diagnostics are not addressed also in the current paper.

2 Related Work

An extensive study of automata models for timed systems is presented in [12]. A general automaton model is defined as the context for developing a variety of simulation proof techniques for timed systems. These techniques include refinements, forward and backward simulations, hybrid forward-backward and backward-forward simulations, and other relations. Relationships between the different types of simulations are stated and proved. To improve model checking performance of timed systems, timing constraint refinement methods such as the efficient forward algorithm based on zones for checking reachability [3] and the counter example guided automatic timing refinement technique [8] have been studied. The refinement of timing has been addressed as part of specification technique recently in [7] where the constructs for refinement of Timed I/O specifications were defined for development of compositional design methodology.

In works [3,7,8] the motivation behind timing refinement has been rather model checking or automated design verification than stepwise transformation-based design development. The way how timing refinement steps are constructed in the course of practical design flow has deserved relatively little attention. For systematic and modular co-use of refinement transformations both data and timing refinement transformations must be specified explicitly in terms of syntactic constraints, i.e., the domain of refinement transformations must consist of well-defined syntactic constructs of the modelling language.

In [4,11,13] attempts have been made to incorporate discrete time directly into formalisms without having a native notion of time. However, the clocks are not an integrated part, but are modelled as ordinary variables. Hence, continuous time specific problems such as Zeno behaviour cannot be addressed directly in, for example, the Event-B proof system. Furthermore, timing can be seen as an extra functional property and adding this to a functional Event-B model will make it cluttered with non-functional aspects. This will, apart from making the model less readable, make the proofs harder to automate.

An earlier attempt to integrate stepwise development in Event-B with model checking in Uppaal is given in [10] where the events are grouped into more coarse grained processes with timing properties. We aim to provide a framework where the timing refinements can be addressed by reusing the model constructs of Event-B introduced in the course of data refinement steps. That provides opportunity to verify data refinement steps also from the timing feasibility point of view.

3 Case-Study: Safety Related Controller Design

The CCD methodology introduced above will be deployed on an industrial redundant safety controller design case study by Danfoss A/S. The case study concerns an emergency shutdown module for a frequency converter that is used to control the speed of an electrical motor. An emergency switch and a safe field bus are used to activate the safety functions. The emergency shutdown module provides two safety functions, the Safe Torque Off (STO) and the Safe Stop 1 (SS1). In this paper we focus on the activation of the safety functions through the emergency switch only. The safety functions are activated if at least one of the two emergency switches (ES) is pushed. STO will remove the torque on the electrical motor. If SS1 is configured active, on activation of SS1 a timer with a configurable delay shall be started and the frequency converter is requested to start a non-safe ramp down. After the timer expires, STO shall be activated. In addition to these functional requirements we need to take into account timing requirements. Specifically, the reaction time from user terminal (pushing of ES) to active STO or active SS1 shall be less than 10 ms. STO or SS1 shall not be activated if the duration of the ES signal is shorter than or equal to 3 ms.

The deployment of our CCD methodology starts with a specification which captures an abstract description of the behaviour of the whole system. Stepwise refinements should introduce the algorithms needed for the implementable system to behave according to the specification. The abstract specification and refinements should be done in such a manner that we can prove all (safety and liveness) properties stated in the requirements as invariant properties or refinements. Because of the space limit, in this paper we will present only one refinement step introducing the needed redundancy. The safety integrity requirements of this safety critical system require the safety controller to be mapped onto a redundant architecture (see 1oo2 architecture of IEC 61508-6 [9]). Before we proceed with the actual modelling of the emergency shutdown module we first present in the next section preliminaries of Event-B and UPTA as well as the mapping between the two formalisms.

4 Preliminaries

4.1 Preliminaries of Event-B

Consider an Event-B model M with variables v, invariant $I(v)$ and events $\mathcal{E}_1, \ldots, \mathcal{E}_m$. All events can be written in the form

$$\mathcal{E}_i = when\, G_i(v)\; then\, v :| S_i(v, v')\; end$$

where $G_i(v)$ is a predicate called the guard and $v :| S_i(v, v')$ is a statement that describes a nondeterministic relationship between variable valuations before and after executing the event. Event-B models do not have a fixed semantics [1], but correctness of a model is defined by a set of proof obligations. We can use these proof obligations to prove correctness of many transition systems. In order to guarantee that $\mathcal{E}_i$ preserves invariant $I(v)$ we need to show that [1]:

- $I(v) \wedge G_i(v) \wedge S_i(v, v') \Rightarrow I(v')$ (INV).

In order to be able to relate Event-B with UPTA in the following sections we interpret an Event-B model as a Labelled Transition System (LTS) $(\Sigma, init, T, i)$, where Σ is the set of states, $init$ is the set of initial states, $T \subseteq \Sigma \times \Sigma$ is the set of transitions and i is the set of legal states $i \subseteq \Sigma$. The set of states g_i where the guard of a transition σ_i holds is given as $g_i = \{v | G_i(v)\}$ and the relation s describing the before after relation for states corresponding to the update statement $v :| S_i(v, v')$ is given as $\{v \mapsto v' | S_i(v, v')\}$. The relation describing the before-after states for each transition T_i is then given as[1] $g_i \lhd s_i$. We can now describe the Event-B model as a transition system $(\Sigma, init, T, i)$ where the state space Σ is formed from the variables $v_1, \ldots, v_n$, $\Sigma = \Sigma_1 \times \ldots \times \Sigma_n$, Σ_i is the type of v_i. The initial states are formed as $init = \{v | Init(v)\}$. The transitions T_i are given as $T_i = g_1 \lhd s_1 \cup \ldots \cup g_m \lhd s_m$. The set of legal states are the ones where the invariant holds $i = \{v | I(v)\}$.

[1] The domain restriction operator $\lhd$ is defined as: $g \lhd s = \{\sigma \mapsto \sigma' \in s | \sigma \in g\}$.

4.2 Preliminaries of UPTA

An UPTA is given 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 allowed in guards. *Sync* is a set of synchronisation actions over channels. An action *send* over a channel h is denoted by $h!$ and its co-action *receive* is denoted by $h?$. *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, $I(Cl, V)$ is the set of invariants over clocks Cl and variables V. $\mathrm{T}_L : L \rightarrow$ {*ordinary, urgent, committed*} is the function that assigns the type to each location of the automaton.

We introduce the semantics of UPTA as defined in [2]. A clock valuation is a function $val_{cl} : Cl \rightarrow \mathbb{R}_{\geq 0}$ from the set of clocks to the non-negative reals. A variable valuation is a function $val_v : V \rightarrow \mathbb{Z} \cup BOOL$ from the set of variables to integers and booleans. Let $\mathbb{R}^{Cl}$ and $(\mathbb{Z} \cup BOOL)^V$be the sets of all clock and variable valuations, respectively. The semantics of an UPTA is defined as an LTS $(\Sigma, init, \rightarrow)$, where $\Sigma \subseteq L \times \mathbb{R}^{Cl} \times (\mathbb{Z} \cup BOOL)^V$ is the set of states, the initial state$init = Init(cl, v)$ for all $cl \in Cl$ and for all $v \in V$, with $cl = 0$, and $\rightarrow \subseteq \Sigma \times \{\mathbb{R}_{\geq 0} \cup Act\} \times \Sigma$ is the transition relation such that:

$$(l, val_{cl}, val_v) \xrightarrow{\mathtt{d}} (l, val_{cl} + \mathtt{d}, val_v) \text{ if } \forall \mathtt{d}' : 0 \leq \mathtt{d}' \leq \mathtt{d} \Rightarrow val_{cl} + \mathtt{d}' \models Inv(l), \text{ and}$$

$$(l, val_{cl}, val_v) \xrightarrow{act} (l', val'_{cl}, val'_v) \text{ if } \exists \mathtt{e} = (l, act, G(cl, v), r, l') \in E \text{ s.t.}$$
$$val_{cl}, val_v \models G(cl, v), val'_{cl} = [re \mapsto 0]val_{cl}, \text{and } val'_{cl}, val'_v \models Inv(l'),$$

where for delay $\mathtt{d} \in \mathbb{R}_{\geq 0}$, $val_{cl} + \mathtt{d}$ maps each clock cl in Cl to the value $val_{cl} + \mathtt{d}$, and $[re \mapsto 0]val_{cl}$ denotes the clock valuation which maps (resets) each clock in re to 0 and agrees with val_{cl} over $Cl \setminus re$.

We have now obtained the correspondence between Event-B and UPTA models through their semantics definition as LTS.

4.3 Mapping from Event-B Models to UPTA

The goal in this paper is to extend the Event-B based stepwise CCD to timed systems. In the subsequent description of CCD model transformations the following notions are used:

- $\mathcal{R}^{\mathcal{T}}$ - Refinement transformation of type $\mathcal{T} \in \{\sqsubseteq_{evt}, \sqsubseteq_{\mathtt{e}}, \sqsubseteq_l\}$, where $\sqsubseteq_{evt}$ stands for *event* refinement, $\sqsubseteq_{\mathtt{e}}$and $\sqsubseteq_l$ stand for *edge* and *location* refinement respectively (elaborated in Section 5).
- $M_i^{\mathtt{T}}$- Model of type $\mathtt{T}$ resulting in i^{th} refinement step, where $\mathtt{T}$ is given as $\mathtt{T} \in \{^B, ^{EFSM}, ^{UPTA}\}$.
- $M_0^{\mathtt{T}}$- The initial specification of type $\mathtt{T}$.
- $\mathtt{T}^{kl} : M^{\mathtt{T}_k} \mapsto M^{\mathtt{T}_l}$ - Syntactic map of model $M^{\mathtt{T}_k}$ to $M^{\mathtt{T}_l}$, where $\mathtt{T}_k$, $\mathtt{T}_l \in \{^B, ^{EFSM}, ^{UPTA}\}$and $\mathtt{T}_k \neq \mathtt{T}_l$.

- $dom(\mathcal{R}^{\mathcal{T}}, M_i^{\mathtt{T}})$ - domain of the refinement $\mathcal{R}^{\mathcal{T}}$ in the model $M_i^{\mathtt{T}}$ (Note that $dom(\mathcal{R}^{\mathcal{T}}, M_i^{\mathtt{T}}) = \emptyset$, if $\mathcal{R}^{\mathcal{T}}$ is not defined in $M_i^{\mathtt{T}}$).
- $ran(\mathcal{R}^{\mathcal{T}}, M_i^{\mathtt{T}})$ - co-domain of the refinement $\mathcal{R}^{\mathcal{T}}$ in the model $M_i^{\mathtt{T}}$.

Let us now state some properties of the transformations:

- For any $M_i^{\mathtt{T}_l} = ran(\mathcal{R}^{\mathcal{T}}, M_i^{\mathtt{T}_k}) \Rightarrow \mathtt{T}_k = \mathtt{T}_l$ ($\mathcal{R}$ is conservative regarding the type of argument model)
- Submodel: $M_j^{\mathtt{T}_i} \subseteq M_i^{\mathtt{T}_i}$ iff $\forall el \in M_j^{\mathtt{T}_i} \Rightarrow el \in M_i^{\mathtt{T}_i}$
- $M_j^{\mathtt{T}} = dom(\mathcal{R}^{\mathcal{T}}, M_i^{\mathtt{T}}) \Rightarrow M_j^{\mathtt{T}} \subseteq M_i^{\mathtt{T}}$, where $dom(\mathcal{R}^{\mathcal{T}}, M_i^{\mathtt{T}})$ is submodel of $M_i^{\mathtt{T}}$, and $M_k^{\mathtt{T}_i}$ is model complement of $M_j^{\mathtt{T}_i}$ in $M_i^{\mathtt{T}_i}$: $M_i^{\mathtt{T}_i} \setminus M_j^{\mathtt{T}_i} = \{el \,|\, el \in M_i^{\mathtt{T}_i},\ el \notin M^{\mathtt{T}_j},\ M_j^{\mathtt{T}_i}\}$
- $M_i^{\mathtt{T}_i} |\, M_j^{\mathtt{T}_j}$- projection of the model $M_i^{\mathtt{T}_i}$ on the model $M_j^{\mathtt{T}_j}$

Due to the fact that $M^B \mapsto M^{UPTA}$ mappings depicted in Fig. 1 preserve locality of M^B changes introduced by refinements, only those model fragments that are introduced by Event-B refinements need to be mapped to the corresponding UPTA fragments. The rest of the refined UPTA model remains same.

The Event-B to UPTA mapping proceeds in following steps:

Step 1: The Event-B model M^B is transformed to a flat Extended Finite State Machine (EFSM) model M^{EFSM} that serves as an UPTA skeleton to be decorated in the next step with UPTA specific timing attributes. The step can be implemented in two sub-steps:

Step 1.1: The Event-B specification is transformed to a Hierarchical Abstract State Transition Machine (HASTM) representation by the algorithm described in [5]. HASTM is a subclass of UML state charts without AND-parallelism.

Step 1.2: The HASTM representation is flattened by means of the algorithm introduced in [6]. The transformation result is an EFSM model M^{EFSM} with operational semantics that is equivalent to the one of the original Event-B model M^B. Thus, the proposed $M^B \mapsto M^{EFSM}$ transformation implements a total injective syntactic map. The derivation of a partially defined (i.e., without timing) UPTA model $M^{UPTA\#}$ from the M^{EFSM} is straightforward:

Let a EFSM model $M^{EFSM} = (\Sigma,\ T,\ V,\ s_0)$, be a syntactic representation of the LTS of the Event-B model M^B, where Σ is a finite set of states, V is a finite set of variables, V may include distinguished subsets I and O - of inputs and outputs respectively, T is the set of transitions, and s_0 is the initial state.

Let $M^{UPTA\#} = (L^{\#},\ E^{\#},\ V,\ \emptyset,\ l_0)$ be an UPTA model without the elements of timing specification, i.e., $L^{\#}$ is a set of locations without clock invariants and committed or urgent types, $E^{\#}$ is a set of edges without clock resets and clock conditions in the edge guards. Then we can establish the correspondence between the elements of M^{EFSM} and $M^{UPTA\#}$ as follows. $L^{\#} = \Sigma$, $E^{\#} = T$, V is the set of variables of the Event-B model M^B (as well as of M^{EFSM} due to **Step 1.2**), $l_0 = s_0$.

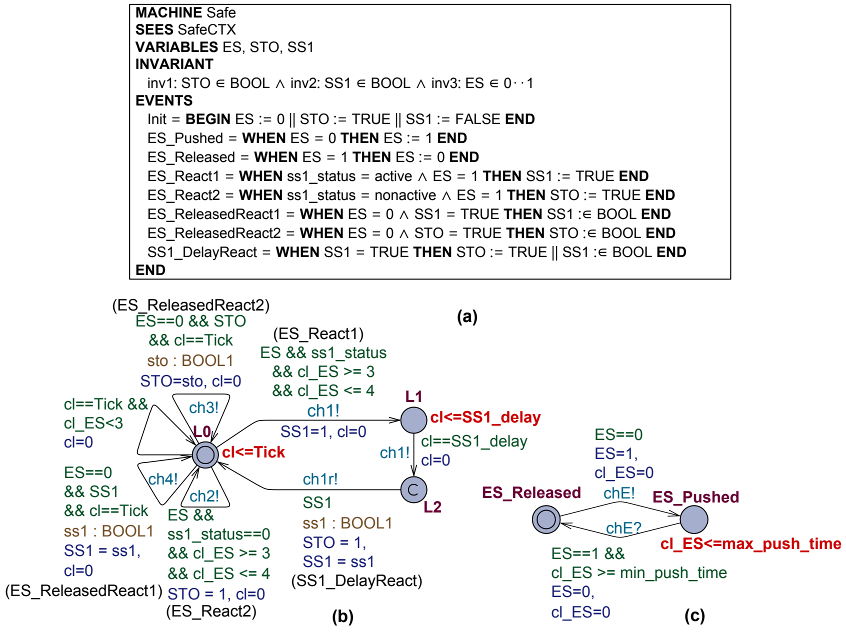
Step 2: In this step the partially defined UPTA model $M^{UPTA\#}$ is extended to full UPTA model M^{UPTA}. The timing constraints to be added to the newly introduced by Event-B refinement $M^{UPTA\#}$ fragment, $M_i^{UPTA\#} = \mathsf{T}^{B,UPTA\#}$ $[ran(\mathcal{R}^T,\ M_{i-1}^B)]$, have to satisfy also the timing constraints of the resultant model M_{i-1}^{UPTA} of the previous timing refinement step. Thus, the arguments of the timing refinement operator $\mathcal{R}_i^T$ to be applied in ith step are untimed $M_i^{UPTA\#}$ and the timing constraints of $dom(\mathcal{R}^T,\ M_{i-1}^B) \mid M_{i-1}^{UPTA})$. The rest of the timing specification M_i^{UPTA}i.e., the parts that do not concern the ith refinement step remain the same as in M_{i-1}^{UPTA}. The formal definition of UPTA timing refinement and its correctness properties are defined in section 5.1.

4.4 Abstract Event-B and UPTA Specifications of Safety Controller

We can now proceed with the formal modelling of the shutdown module exemplifying the mapping from Event-B to UPTA proposed in the previous subsection. We start the modelling by defining only three externally observable variables, namely *ES*, *STO* and *SS1*, in the abstract Event-B model. The abstract Event-B specification M_0^B named *Safe* is presented in Fig. 2a. We have a configuration parameter defined in *SafeCTX* named *ss1_status* which can be set either to *active* or *nonactive*. If *ss1_status* is set to *active* then activation of *STO* should occur after the delay when *SS1* is activated. All the variables of this abstract model are of the boolean type *BOOL*. Value *TRUE* corresponds to *STO* or *SS1* being active and value *FALSE* corresponds to *STO* or *SS1* being nonactive.

At initialisation *STO* is active. Event *ES_ReleasedReact2* takes care of reseting *STO*. Events *ES_Pushed* and *ES_Released* separately model the physical act of pushing and releasing *ES* and they can be considered as input events from the environment to the system. Events *ES_React1* and *ES_React2* model the eventual reaction of the system to the pushing of *ES*; *ES_React1* corresponds to the case when *ss1_status=active* and *ES_React2* corresponds to the case when *ss1_status=nonactive*. Similarly, we distinguish between these two cases for the reaction of the system to the release of *ES* with events *ES_ReleasedReact1* and *ES_ReleasedReact2*. Already at this abstraction level we need to consider the redundancy of the system. Thus, these events are non-deterministic because both redundant systems need to first react to the pushing of *ES* and then they can be disabled. Moreover, the redundant STO outputs are disabled asynchronously. The last event *SS1_DelayReact* models the activation of *STO* after the timer triggered by the activation of *SS1* has expired. For the same reasons as above this event is non-deterministic too at this point. The initial model describes the desired functionality of the system in such a manner that is easy to get an overview of the intended behaviour.

Let us now introduce the corresponding UPTA abstract specification resulting from M_0^B *Safe* incorporating the timing requirements. We map the events of M_0^B *Safe* to edges of M_0^{UPTA} that is defined as parallel composition of automata *Safe* and *Environment*. The parallel composition of automata is needed to avoid explicit modelling of system and environment events' interleaving. Similarly, interleaving of system and environment events is implicit in Event-B. Events *ES_Pushed* and *ES_Released* of M_0^B *Safe* are modelled in automaton M_0^{UPTA} *Environment* illustrated in Fig. 2c and all other events of M_0^B *Safe* in



```
MACHINE Safe
SEES SafeCTX
VARIABLES ES, STO, SS1
INVARIANT
  inv1: STO ∈ BOOL ∧ inv2: SS1 ∈ BOOL ∧ inv3: ES ∈ 0‥1
EVENTS
  Init = BEGIN ES := 0 || STO := TRUE || SS1 := FALSE END
  ES_Pushed = WHEN ES = 0 THEN ES := 1 END
  ES_Released = WHEN ES = 1 THEN ES := 0 END
  ES_React1 = WHEN ss1_status = active ∧ ES = 1 THEN SS1 := TRUE END
  ES_React2 = WHEN ss1_status = nonactive ∧ ES = 1 THEN STO := TRUE END
  ES_ReleasedReact1 = WHEN ES = 0 ∧ SS1 = TRUE THEN SS1 :∈ BOOL END
  ES_ReleasedReact2 = WHEN ES = 0 ∧ STO = TRUE THEN STO :∈ BOOL END
  SS1_DelayReact = WHEN SS1 = TRUE THEN STO := TRUE || SS1 :∈ BOOL END
END
```

Fig. 2. (a) Event-B model M_0^B *Safe*, (b) UPTA model M_0^{UPTA} *Safe* and (c) UPTA model M_0^{UPTA} *Environment*

automaton M_0^{UPTA} *Safe* illustrated in Fig. 2b. Since the only causally dependent event pair in M_0^B is *ES_React1* and *SS1_DelayReact* their sequencing is mapped in M_0^{UPTA} to the pair of edges connected via location $L1$. Second reason for$L1$ is to model *SS1_delay*. $L2$ is introduced for technical reasons to limit the number of channels per edge in UPTA. In fact, channels depicted in Fig. 2b and 2c are obsolete for M_0^{UPTA}, they are shown only because of specifying synchronization constraints with the edges of refined model M_1^{UPTA} in Fig. 4b and 4c respectively.

The timing constraints added to M_0^B events are specified in M_0^{UPTA} by means of model clocks, clock guards and clock resets. To avoid the interference of clock constraints of parallel automata there is one local clock defined for each automaton. An extra local clock is needed only when there is simultaneously timeout bounded waiting for an external event and periodic time bound actions executed during that timeout period. For instance, the invariant $cl \leq Tick$ of location $L0$ guarantees that each time after the state variables are updated some of the enabled transitions in $L0$ will be executed latest at time instant specified by constant *Tick*. The clock guard $cl == Tick$ of edge *Idle time pass* ensures that if there is not any transition enabled within *Tick* then at least the clock *cl* reset action has to be taken by executing edge *Idle time pass* with update $cl = 0$. The alternative clock *cl_ ES* is needed for specifying the timing of simultaneous events *ES_ Pushed* and *ES_ Released*. One clock can be used also for specifying alternative time delays when used at different locations, e.g. the invariant

$cl \leq SS1\_delay$ of location $L1$ and clock guard $cl == SS1\_delay$ of edge outgoing $L1$ model the SS1 delay.

5 Proving Refinement of Timed Systems

We now have an interpretation of Event-B models as UPTA. The goal is to ensure correctness of the abstract model and its refinements. Let us first introduce the refinement definition for timed systems presented by Lynch and Vaandrager [12]. We adapt this definition in order to correspond to the UPTA semantics of section 4.2. Let cl_c and cl_a be the concrete and abstract clocks of refinement and specification model N and M respectively.

Definition 1. *A specification M is refined by a specification N, written $M \sqsubseteq N$, iff there exists a binary relation $\mathsf{R} \subseteq \Sigma^N \times \Sigma^M$ such that for each pair of states $(n, m) \in \mathsf{R}$ we have:*

1. whenever $n_{(l_{ref}, val_{cl_c}, val_w)} \xrightarrow{act^N} n'_{(l'_{ref}, val'_{cl_c}, val'_w)}$ for some $n' \in \Sigma^N$ then $m_{(l_{abs}, val_{cl_a}, val_v)} \xrightarrow{act^M} m'_{(l'_{abs}, val'_{cl_a}, val'_v)}$ and $(n', m') \in \mathsf{R}$ for some $m' \in \Sigma^M$

2. whenever $n_{(l_{ref}, val_{cl_c}, val_w)} \xrightarrow{\mathtt{d}^N} n'_{(l_{ref}, val_{cl_c}+\mathtt{d}, val_w)}$ for $\mathtt{d} \in \mathbb{R}_{\geq 0}$ then $m_{(l_{abs}, val_{cl_a}, val_v)} \xrightarrow{\mathtt{d}^M} m'_{(l_{abs}, val_{cl_a}+\mathtt{d}, val_v)}$ and $(n', m') \in \mathsf{R}$ for some $m' \in \Sigma^M$

Let us introduce the original Event-B proof obligations that are considered in this paper and needed to be discharged in order to ensure correctness of refinements. Let M and N be Event-B models, where $M \sqsubseteq N$ with refinement transformation $\mathcal{R}^{\mathcal{T}}$ of type $\mathcal{T} \in \{\sqsubseteq_{evt}\}$, where $\sqsubseteq_{evt}$ stands for event refinement. Let w be the concrete variables of model N and $J(v, w)$ be the concrete invariant stating properties on variables w and the gluing invariant between the abstract and concrete state space. For a concrete transition r of model N with guards $H(w)$ and before-after predicate $R(w, w')$ refining the abstract transition e defined in section 4.1 the following proof obligations [1] need to be discharged:

– $I(v) \wedge J(v, w) \wedge H(w) \Rightarrow G(v)$ (GRD),
– $I(v) \wedge J(v, w) \wedge H(w) \wedge R(w, w') \Rightarrow \exists v'. S(v, v') \wedge J(v', w')$ (INV-SIM),

where obligation GRD states the guard strengthening of event e by event r, obligation INV-SIM states the preservation of the invariant after updates on variables v and w and the simulation of the abstract event e by the concrete event r. It is easy to see that condition 1 of Definition 1 can be checked by the Event-B proof obligations given above [1]. Since the UPTA models are constructed from the Event-B models we need to extend the original refinement proof obligations GRD and INV-SIM to also check the timing obligations of all conditions of Definition 1.

Let $J_t(cl_a, cl_c)$ be the concrete invariant stating properties on abstract and concrete clocks cl_a and cl_c in a timed system. Based on the untimed Event-B proof obligations given above we propose the following proof obligations elaborated with timing conditions for a concrete event r of model N with guards

$H(w)$ and $H_t(cl_c)$, before-after predicate $S(w, w')$ and clocks reset on cl_c refining the abstract event e defined in section 4.1:

- $I(v) \wedge J(v, w) \wedge I_t(cl_a) \wedge J_t(cl_a, cl_c) \wedge H(w) \wedge H_t(cl_c) \Rightarrow G(v) \wedge G_t(cl_a)$ (GRD+),
- $I(v) \wedge J(v, w) \wedge H(w) \wedge R(w, w') \wedge I_t(cl_a) \wedge J_t(cl_a, cl_c) \wedge H_t(cl_c) \wedge cl_c' = 0 \Rightarrow \exists v', cl_a' \cdot S(v, v') \wedge J(v', w') \wedge cl_a' = 0 \wedge J_t(cl_a', cl_c')$ (INV-SIM+).

Note that if there are no clock resets on the event the clock reset updates are omitted. Let us elaborate more on obligation GRD+. Invariant $J_t(cl_a, cl_c)$ can be defined as:

- $J_t(cl_a, cl_c) = J1_t(cl_a, cl_c) \wedge J2_t(cl_c)$,

where $J1_t$ is the gluing invariant expressing that (i) resets of cl_a and cl_c are synchronous when entering source locations $pre(e)$ and $pre(r)$ of events e and r, respectively, and (ii) $J_t(cl_c) \wedge H_t(cl_c) \Rightarrow I_t(cl_a) \wedge G_t(cl_a)$. $J2_t$ is the invariant of the source location of r.

The proposed INV-SIM+ proof obligation can be decomposed by logic rules to two substatements (for each pair of concrete and abstract clocks $cl_c$and cl_a):

- $I(v) \wedge J(v, w) \wedge H(w) \wedge R(w, w') \Rightarrow \exists v' \cdot S(v, v') \wedge J(v', w')$,
- $I_t(cl_a) \wedge J_t(cl_a, cl_c) \wedge H_t(cl_c) \Rightarrow J_t(0, 0)$.

The first condition corresponds to the original proof obligation within Event-B, while the second condition is the correctness condition for resetting of clocks. The general functional and timing statements introduced in this subsection are instantiated by UPTA syntax related refinement conditions introduced in the next subsection.

5.1 Superposition Refinement of UPTA

To check the timing refinement conditions in UPTA, we use an approach where the abstract UPTA model is composed in parallel with models that describe the refined parts of the abstract model. Thus, the refinements are added to the abstract model incrementally in the course of design development process and to support compositional solving of model checking tasks. For composition the refinements have a wrapping construct "context frame" that allows their uniform and easy injection into the abstract model. To keep the clear correspondence between syntactic units of the abstract model and refinement we define the refinement transformations syntactic element wise, i.e., by locations and edges of UPTA calling them *location* and *edge* refinements respectively.

Also, the course of refinement is made explicit in the model by introducing each refinement step as an increment to the resultant model of the previous refinement. It means that without changing the semantics of the abstract model we add the refinement of its syntactic element el as new automaton M^{el} that is composed with the original model M. For composition we use synchronized parallel composition $\|_{sync}$, i.e. $M \sqsubseteq M \|_{sync} M^{el}$. Synchronization of M and M^{el} is needed to preserve the contract of *el* with its context after refinement. Technically, it means decorating the primary automaton M with auxiliary channel labels to synchronize the entry and leave points to/from the element *el* of M.

For further elaboration of the technique, we define *location refinement* ($\sqsubseteq_l$) and *edge refinement* ($\sqsubseteq_{\mathtt{e}}$) relations separately. Notice that the *event* (*guard* and *update*) *refinement* introduced in Event-B [1] can be considered as special case of edge refinement where the guards of edges are strengthened and new updates are added respectively in the refining model $M^{\mathtt{e}}$ consisting of exactly one edge.

Edge Refinement. We say that a synchronous parallel composition of automata M and $M^{\mathtt{e}_i}$ is an edge refinement for edge $\mathtt{e}_i$ of M, ($M \sqsubseteq_{\mathtt{e}} M \| M^{\mathtt{e}_i}$) iff:

$$\mathtt{e}_i \in E(M), \text{ and there exists } M^{\mathtt{e}_i} \textit{ such that } P_1 \wedge P_2 \wedge P_3 \wedge P_4,$$

where P_1(*interference free new updates*): No variable of M is updated in $M^{\mathtt{e}_i}$, i.e. no variable of M occurs in the left-hand side of any update in $M^{\mathtt{e}_i}$.

- P_2(*guard splitting*): Let $<l_0^{'}, l_F^{'}>$ denote a set of all feasible paths from the initial location $l_0^{'}$ to final location $l_F^{'}$ in $M^{\mathtt{e}_i}$ and $<l_0^{'}, l_F^{'}>_k \in <l_0^{'}, l_F^{'}>$ be k^{th}path in that set. Then,
  - $\forall k \in [1, | < l_0^{'}, l_F^{'} > |]. \wedge_{j \in [1, Length(k)]} G(\mathtt{e}_j^{'}) \Rightarrow G(\mathtt{e}_i)$,

i.e., the conjunction of edge guards of any path in $<l_0^{'}, l_F^{'}>$ is not weaker than the guard of the edge $\mathtt{e}_i$ refined.

- P_3(0-*duration unwinding*): $\forall l_i^{'} \in (L_{M^{\mathtt{e}_i}} \setminus l_0^{'}). \mathrm{T}(l_i^{'}) = \textit{committed}$,

i.e., all edges in the refinement $M^{\mathtt{e}_i}$ must be atomic and locations *committed.*

- P_4(*non-divergency*): $G(\mathtt{e}_i) \Rightarrow M^{\mathtt{e}}, l_0^{'} \models \mathtt{A}\Diamond l_F^{'}$,

i.e., validity of $G(\mathtt{e}_i)$ implies the existence of a feasible path in $M^{\mathtt{e}_i}$.

The context frame needed to implement the edge refinement is depicted in Fig. 3a. It includes auxiliary locations $l_0^{'}$ and $l_F^{'}$.

Location Refinement. We say that a synchronous parallel composition of automata M and M^{l_i} is a location refinement for location l_i of M, ($M \sqsubseteq_l M \| M^{l_i}$) iff $l_i \in L_M$, and exists M^{l_i} s.t. $P_1^{'} \wedge P_5 \wedge P_4^{'}$, where:

- $P_1^{'}$(*interference free new updates* - same as P_1 above).
- P_5(*preservation of non-blocking invariant*):
  - $[(M \| M^{l_i}), (l_0, l_0^{'}) \models \mathtt{E}\Diamond \texttt{deadlock}] \Rightarrow [M, l_0 \models \mathtt{E}\Diamond \texttt{deadlock}]$.
- $P_4^{'}$(*non-divergency*):
  - $inv(l_i) \equiv x \leq \mathtt{d}$ for $x \in Cl_M$, $\mathtt{d} < \infty \Rightarrow [M^{l_i}, l_0^{'} \models l_0^{'} \leadsto_{\mathtt{d}} l_F^{'}]$,

where "$\rightsquigarrow_{\mathrm{d}}$" denotes bounded reachability operator with non-negative integer time bound, Cl_M is the set of clocks of M, locations $l_0^{'}$ and $l_F^{'}$ denote respectively auxiliary pre- and post-locations in the context frame of the refinement.

P_5 and $P_4^{'}$ are specified as Uppaal model checking queries expressed in TCTL. "*deadlock*" denotes a standard predicate in Uppaal about the existence of deadlocks in the model. $P_4^{'}$ requires that the invariant of l_i is not violated due to accumulated delays of M^{l_i} runs. A graphical representation of the model fragment that schematically represents location refinement is depicted in Fig. 3b.

The auxiliary context frame of $\sqsubseteq_l$ consists of the following elements (denoted by dashed line in Fig. 3b):

– Synchronizing channel ch is needed to synchronize the executions of entering to and departing from location l_i transitions with those entering and departing to and from refining model M^{l_i}.
– Auxiliary initial location l_0' and final location $l_F^{'}$ of M^{l_i} are introduced to model waiting before the synchronization via channel ch arrives and after the execution of M^{l_i} terminates.

Location refinement can be applied when the refinement M^{l_i} specifies non-instantaneous time bounded behaviours that are represented in abstract model as location l_i.

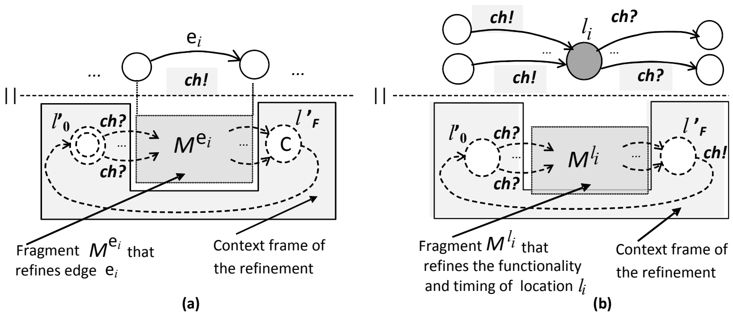

e_i
...
ch!
...
l'0
ch?
ch?
M^{e_i}
C
l'F
Fragment M^{e_i} that refines edge e_i
Context frame of the refinement
(a)
ch!
l_i
ch?
ch!
ch?
l'0
ch?
ch?
M^{l_i}
l'F
ch!
Fragment M^{l_i} that refines the functionality and timing of location l_i
Context frame of the refinement
(b)

Fig. 3. (a) A fragment of the primary model M and refinement $M^{\mathsf{e}_i}$of edge e_i. (b) A fragment of the abstract model M with location l_i and its refinement M^{l_i}.

5.2 Event-B and UPTA Refinement of Safety Controller

Let us now exemplify the refinement theory proposed above by performing a refinement on the abstract safety controller specification. The Event-B refinement presented in this subsection introduces the required redundancy for the system. The excerpt of the Event-B refined model M_1^B *Safe1* is presented in Fig. 4a. Each (redundant) variable becomes a function from the set of CPUs (including two instances, *cpu1* and *cpu2*) to some boolean value. The new set is specified in a refined context. The refined variables are named *ES_Redundant*, *STO_Redundant* and *SS1_Redundant*.

```
MACHINE  Safe1
REFINES  Safe
SEES  SafeCTX, RedundancyCTX
VARIABLES  ES_Redundant, STO_Redundant, SS1_Redundant
INVARIANT
 inv1: STO_Redundant ∈ CPUS → BOOL ∧ ... ∧
 inv4: STO = FALSE ⇔  (STO_Redundant(cpu1)=FALSE ∧
                              STO_Redundant(cpu2)=FALSE) ∧
 inv5: STO = TRUE ⇔ (STO_Redundant(cpu1)=TRUE ∨
                              STO_Redundant(cpu2)=TRUE) ∧...
VARIANT  ...
EVENTS
  Init = ...
  ES_Pushed =
    ANY ES1, ES2 WHERE
      ES1, ES2 ∈0..1 ∧ ES_Redundant(cpu1) = 0 ∧
      ES_Redundant(cpu2) = 0 ∧ ¬ (ES1=0 ∧ES2=0)
    THEN ES_Redundant := {cpu1↦ES1, cpu2↦ES2} END
  ES_Released = ...
  ES_React1 =
    ANY cpu WHERE
      cpu ∈CPUs ∧ss1_status = active ∧ ES_Redundant(cpu) = 1 ∧
      SS1_Redundant(cpu) = FALSE ∧STO_Redundant(cpu) = FALSE
    THEN SS1_Redundant(cpu) := TRUE END
  ES_React2 = ...
  ES_ReleasedReact1 =
    ANY cpu WHERE
      cpu ∈CPUs ∧ES_Redundant(cpu1) = 0 ∧
      ES_Redundant(cpu2) = 0 ∧SS1_Redundant(cpu) = TRUE
    THEN SS1_Redundant(cpu) := FALSE END
  ES_ReleasedReact2 = ...
  ES_DelayReact = ...
END
```

(a)

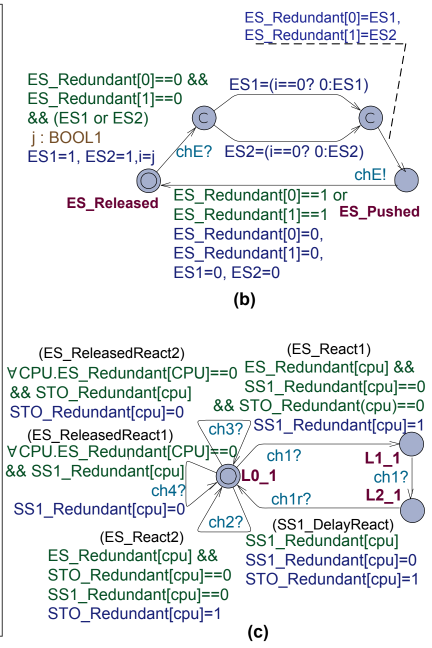


Fig. 4. (a) Event-B refinement $M_1^B Safe1$, (b) UPTA refinement M_1^{UPTA} *Safe1* and (c) UPTA model M_0^{UPTA} *Environment*

Some gluing invariants are required to relate the abstract state with the more concrete state. Considering the deactivation and activation of *STO*, we need two different invariants. If *STO* is deactivated, then none of the redundant outputs are activated (*inv4* in Fig. 4a). If *STO* is activated, then at least one of the redundant outputs is activated (*inv5* in Fig. 4a). The reason for such invariants is that it is enough for only one of the redundant inputs and outputs to work in order for the system to be safe. Similar pairwise gluing invariants exist for the other two redundant variables.

Let us focus on some of the refined events. Event *ES_ Pushed* is refined as follows in order to model the redundant pushing of *ES*. Either *ES* can activate its corresponding safety function. The refined events *ES_ React1* and *ES_ React2* model the reaction of each CPU to its corresponding *ES*. The refinement of the previously non-deterministic events *ES_ ReleasedReact1* and *ES_ ReleasedReact2* takes into account the redundant CPUs. In order to allow reseting of an activated redundant safety function it is required that both redundant *ES* are released (handled by event *ES_ Released*). This can handle failure of an *ES* which would cause the deactivation of its corresponding safety function.

Since only new variables and their updates are introduced by Event-B refinement $\mathcal{R}^{\sqsubseteq_{evt}} : M_0^B \rightarrow M_1^B$ the mapping $\mathsf{T}^{B,UPTA\#} : M_1^B \mapsto M_1^{UPTA\#}$ copies the control structure of model $M_0^{UPTA\#}$ by introducing two structurally identical parallel instances $M_1^{UPTA(0)}$ and $M_1^{UPTA(1)}$ to model the redundancy. Note that both instances need their own context frame. Technically, copying the control structure of M_0^{UPTA}in M_1^{UPTA}can be considered as an aggregate model resulting from refinement steps of all edges with simple variable renaming. Since the timing of M_1^{UPTA} does not differ from that of M_0^{UPTA} the edge *Idle time pass* and location $L2$ of type *committed* added to $M_0^{UPTA\#}$ when specifying timing, do not need duplication. The edge refinement preserves the timing behaviour of M_0^{UPTA} by construction. Regardless the aggregation of several parallel refining models $M_1^{UPTA(i)}$into one in Fig. 4b, the synchronization defined by $\mathcal{R}^{\sqsubseteq_e}$needs to be preserved between the edges of M_0^{UPTA} and their refinements in M_1^{UPTA}.

The correctness of $\mathcal{R}^{(\sqsubseteq_e)} : M_0^{UPTA} \rightarrow M_1^{UPTA}$ follows trivially from the proof obligations of the definition of edge refinement given in Section 5.1. Both the consistency of abstract timing specification of $M_0^{UPTA}$and the correctness of timing refinement are verified by means of the Uppaal model checker. The proof obligations of this refinement are expressed in TCTL (query language of Uppaal). For instance, the reaction time requirement in M_0^{UPTA} is expressed as bounded (with time bound t) liveness property $\varphi \rightarrow_{\leq t} \psi$. The query:

– $(ES \,\&\&\, cl\_ES \geq 3) \rightarrow ((STO \text{ } or \text{ } SS1) \,\&\&\, gclock \leq 10)$

is satisfied for M_1^{UPTA} if and only if after starting pushing *ES* longer than 3 time units the state where *STO* or *SS1* is active is reached always within 10 time units. After timing refinement (in our example applying edge refinement as described above) one needs to model check the properties P_2 and P_4. For P_2 it suffices from checking implication between guards of refined and abstract model edges. Checking P_4 reduces to checking the queries of form $\mathtt{A}\Diamond post(r)$ where $post(r)$ denotes the post location of edge r in refined model M_1^{UPTA}. Properties P_1 and P_3 are subject to simple syntactic checks.

6 Conclusion and Future Work

We propose a correct-by-construction design workflow where model-based design transformations combine alternating data and timing constraints refinement steps. The goal is to benefit from mutually complementing formalisms Event-B and Uppaal automata and related verification techniques. For bridging the data and timing refinement steps the Event-B to UPTA map and its timing refinement transformations have been defined. That allows to verify the data refinement correctness also from its timing feasibility point of view. The approach is demonstrated on a fragment of an industrial case study of a safety critical system. The approach does not guarantee the fully incremental design flow, backtracking is needed when there is no feasible timing refinement possible for a given data refinement result. The design backtracking and error diagnostics are not addressed in the current paper. Also the automation of the proposed design transformations remain for future work.

Acknowledgement. This work has been partially funded by RECOMP project within the ARTEMIS joint undertaking (Grant agreement no. 100202). We would also like to thank the reviewers for their useful comments.

References

1. Abrial, J.-R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press (2010)
2. Behrmann, G., David, A., Larsen, K.G.: A Tutorial on Uppaal. In: Bernardo, M., Corradini, F. (eds.) SFM-RT 2004. LNCS, vol. 3185, pp. 200–236. Springer, Heidelberg (2004)
3. Bouyer, P., Laroussinie, F., Reynier, P.-A.: Diagonal Constraints in Timed Automata: Forward Analysis of Timed Systems. In: Pettersson, P., Yi, W. (eds.) FORMATS 2005. LNCS, vol. 3829, pp. 112–126. Springer, Heidelberg (2005)
4. 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)
5. Chaudhari, D.L., Damani, O.P.: Generating hierarchical state based representation from Event-B models. In: Proceedings of the B 2011 Workshop. ENTCS, vol. 280 (2011)
6. Chimisliu, V., Wotawa, F.: Abstracting timing information in UML state charts via temporal ordering and LOTOS. In: AST 2011, pp. 8–14. ACM (2011)
7. David, A., Larsen, K.G., Legay, A., Nyman, U., Wasowski, A.: Timed I/O Automata: A complete specification theory for real-time systems. In: HSCC 2010. ACM (2011)
8. Dierks, H., Kupferschmid, S., Larsen, K.G.: Automatic Abstraction Refinement for Timed Automata. In: Raskin, J.-F., Thiagarajan, P.S. (eds.) FORMATS 2007. LNCS, vol. 4763, pp. 114–129. Springer, Heidelberg (2007)
9. International Electrotechnical Commission (IEC). IEC 61508-6: Functional safety of electrical/electronic/programmable electronic safety-related systems, 2nd edn. (2010)
10. Iliasov, A., Laibinis, L., Troubitsyna, E., Romanovsky, A., Latvala, T.: Augmenting Event-B modelling with real-time verification. Technical Report 1006, TUCS (2011)
11. Lamport, L.: Real-Time Model Checking Is Really Simple. In: Borrione, D., Paul, W. (eds.) CHARME 2005. LNCS, vol. 3725, pp. 162–175. Springer, Heidelberg (2005)
12. Lynch, N., Vaandrager, F.: Forward and backward simulations - Part II: Timing-Based systems. Information and Computation 128 (1995)
13. Sarshogh, M.R., Butler, M.: Specification and refinement of discrete timing properties in Event-B. Technical report, Electronic and Computer Science, University of Southampton (2011)