Proving timing properties with the Leibnizian time model

Alexei Iliasov

Newcastle University

Abstract. We present a novel approach to the description of real-time requirements in Event-B, based on the relativistic time model of Gottfried Liebniz. The approach is surprisingly useful, and has led to some significant results. We illustrate the approach with several modelling recipes for the specification of real-time systems in Event-B.

1 Introduction

In the design and modelling of systems from user specifications, it is common to find some proportion of the user requirements expressed in terms of real-time. In work on business information systems, for example, real-time requirements are a natural way to express high-level constraints on business processes [8]. In scheduling or performance analysis, real-time is the natural language for stating requirements.

We use the Event-B language [5] to explore an alternative model of time, the Leibnizian model [11]. According to Leibniz, time is not a fundamental dimension, but is used to distinguish the changes in an observed entity. In the Newtonian model time is an observable attribute of an entity, and may be used to distinguish an entity in the past from an entity in the future, even if the entities are otherwise identical. In the Leibnizian model, in which time is not a directly observable attribute, these may only be distinguished if some other observable attribute has changed. In other words, in the Leibnizian model, time-related changes are transformations of the entity itself. If nothing changes, time is not observed to pass, and therefore (to the observer) time does not pass. The Newtonian model permits time to change without a change in the observed entity.

The dichotomy of the Leibnizian model, in which two separate entities are necessary in order to define the notion of time, suggests that all the time-related properties may be isolated in the observer part leaving the part being observed to deal with functional properties. This has important practical implications: the formulation of timing constraints does not have to be notationally tied with the description of behaviour so that existing methods, semantics and tools may be employed in specifying functional properties.

This difference in time interpretation has significant consequences for the definition of a timed semantics, and for the specification of timing constraints in Event B. We present the semantic model briefly, using examples to illustrate the

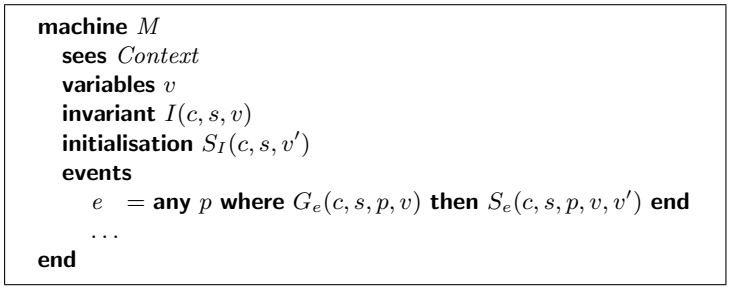

Fig. 1. Event-B model structure.

important points. We also give a series of "recipes" to show how the Leibnizian time model could be used by a model developer to introduce time into Event-B developments.

2 Background

An Event-B development starts with a compact, often trivial abstraction. The cornerstone of the Event-B method is a stepwise development that facilitates a gradual design of a complex system via a number of correctness-preserving *refinement* steps. The general form of an Event-B model (or *machine*) is shown in Fig. 1. A machine encapsulates a state space, defined by *machine variables*, and provides transitions on the state, as described by *machine events*. Events are characterised by a list of parameters p, a state predicate G called an *event guard*, and a next-state relation S.

The **invariant** clause defines the properties of a system, expressed as state predicates, that must be preserved during the system lifitem. The states defined by an invariant are called the *safe states* of a system. A correct model is proven to never leave its safe states. Data types s, constants c and relevant axioms are defined in a separate component called a *context*, and included into a machine with the **sees** clause.

The consistency of a machine as well as the correctness of refinement steps is demonstrated by discharging relevant *proof obligations* which, collectively, define the Event-B *proof semantics* [5]. The Rodin Platform [18], a tool supporting Event-B, is an integrated environment that automatically generates necessary proof obligations and providers a number of automated provers and solvers along with an interactive proof environment.

An Event-B machine defines a state transition system. Let $\Omega = \{v \mid I(c, s, v)\}$ be the (safe) states of a machine where v and $I(c, s, v)$ are the variables and the invariant of a machine. The relational form of an event e is $[e]_R \equiv \{v \mapsto v' \mid \exists p \cdot (G_e(c, s, p, v) \wedge S_e(c, s, p, v, v'))\}$.

Definition 1 (Event-B transition system). *A machine defines a transition system* (Ω, f, ω_0) *where* $f : \Omega \rightarrow \mathbb{P}(\Omega)$ *is defined as* $f = (\bigcup_e [e]_R)$*; the set of initial states* $\omega_0 \subseteq \Omega$ *is defined by the initialisation predicate* S_I*:* $\omega_0 = \{v' \mid S_I(c, s, v')\}$.

3 Leibnizian Time

In this section we formally define some essential concepts of the Leibnizian time model. We illustrate them with a timed specification of a lossless buffer, which we return to throughout the paper. For brevity, we omit the theorem proofs. Proofs and machine-checked models of the example are available at [3].

A fundamental concept is that of a process, which we define as a transition system.

Definition 2 (Process). *A process* P *is a tuple* $(\alpha\mathsf{P}, p, \iota\mathsf{P})$ *where* $\alpha\mathsf{P}$ *is a process* alphabet*,* $p \subset \alpha\mathsf{P} \times \alpha\mathsf{P}$ *is a* transition relation *and* $\iota\mathsf{P}$ *is the set of initial states.*

Time only appears when we put together two processes and let them interact in a certain way. The nature of the interaction is what intuitively may be regarded as an observation of one process by another.

Definition 3 (Observation connection). *An* observation connection *between processes* C *and* S *is a relation* $\varphi \subseteq \alpha\mathsf{S} \times \alpha\mathsf{C}$.

A timed system is formed of pair of processes where one process, an *observer*, is said to observe another process, a *subject*. In the definition above, C is an observer and S is a subject.

Definition 4 (Timed system). *An observer process* C*, a subject process* S *and an observation connection* φ *define a* timed system $\mathsf{C} \cdot \varphi \cdot \mathsf{S}$.

The first technique we give extends an untimed Event-B model to a timed system, by defining a timed observer in an associated context. We illustrate this technique in Example 1.

Recipe 1 (Event-B timed system) An timed Event-B system $\mathsf{C} \cdot \varphi \cdot \mathsf{S}$ is a pair of a machine S and context C of the following form.

machine S
  sees C
  variables v
  invariant $I(V)$
  initialisation $R(v')$
  events
    $E_i =$ **any** p_i **where**
        $G_i(p_i, v)$
      then
        $S_i(p_i, v, v')$
      end
end

context C
  sets $\alpha\mathsf{C}$
  constants $c, \varphi, \iota\mathsf{C}$
  axioms
    $\iota\mathsf{C} \subseteq \alpha\mathsf{C}$
    $c \subseteq \alpha\mathsf{C} \times \alpha\mathsf{C}$
    $\varphi \subseteq \{v \mid I(v)\} \times \alpha\mathsf{C}$
    $\ldots$
end

Subject S is an arbitrary Event-B machine defining a vector of variables v. Set $\{v \mid I(v)\}$ defines the possible states of the machine. Observer C is axiomatically defined in a context. The context defines a sort $\alpha\mathsf{C}$, a transition relation c and an observation connection φ which relates states from set $\{v \mid I(v)\}$ to observer states. Further axioms and theorems may added, to more precisely characterise the observer model. □

Example 1 (Buffer). A *lossy* buffer with the capacity to store one element of type V is defined by machine BUF, as shown below.

machine BUF
  sees $def, \mathsf{C_0}$
  variables b
  invariant $b \in V$
  initialisation $b :\in V$
  events
    $wr =$ **any** v **where**
        $v \in V1$
      then
        $b := v$
      end
    $rd \ =$ **begin** $b := nil$ **end**
end

context $\mathsf{C_0}$
  $\iota\mathsf{C_0} = V$
  $c \subseteq V \times V \setminus (V1 \times V1)$
  $\varphi = V \lhd \mathrm{id}$
end

The constant $nil \in V$ and sets $V1 = V \setminus \{nil\}$, $V1 \neq \varnothing$ are defined in context def. Event wr updates the value of the stored element; event rd consumes a buffered element and sets the buffer contents to nil to indicate that the buffer is now empty. The events are always enabled and thus BUF permits arbitrary interleavings of the operations. Such operations may be implemented by unsynchronised concurrent activities. The write operation may happen arbitrary often thus potentially overwriting a previous value before it is read.

A *lossless* buffer is defined with the following timed Event-B system.

$$\mathsf{C_0} \cdot \varphi \cdot \mathsf{BUF}$$

The observation model rules out the possibility of event wr writing into a non-empty buffer. We shall substantiate this claim in Example 2. □

An interpretation of a timed system gives a precise meaning to the phenomenon of observation. Essentially, an observation prohibits behaviours that an observer does not expect to see.

Definition 5 (Interpretation of a timed system). *Given a timed system* $\mathsf{C} \cdot \varphi \cdot \mathsf{S}$ *where* $\mathsf{S} = (\alpha\mathsf{S}, s, \iota\mathsf{S})$ *and* $\mathsf{C} = (\alpha\mathsf{C}, c, \iota\mathsf{C})$*, its interpretation is a process*

$$\mathbb{I}(\mathsf{C} \cdot \varphi \cdot \mathsf{S}) \equiv (\varphi, \tau(\mathsf{C} \cdot \varphi \cdot \mathsf{S}), (\iota\mathsf{S} \times \iota\mathsf{C}) \cap \varphi)$$

where transition relation $\tau(\mathsf{C} \cdot \varphi \cdot \mathsf{S}) \subseteq (\alpha\mathsf{S} \times \alpha\mathsf{C}) \times (\alpha\mathsf{S} \times \alpha\mathsf{C})$ *is such that a mapping* $(u \mapsto t) \mapsto (u' \mapsto t') \in (\alpha\mathsf{S} \times \alpha\mathsf{C}) \times (\alpha\mathsf{S} \times \alpha\mathsf{C})$ *belongs to* $\tau(\mathsf{C} \cdot \varphi \cdot \mathsf{S})$ *if and only if the following properties hold*

(a) $u \mapsto u' \in s$ *(a transition of a subject process)*
(b) $t \mapsto t' \in c$ *(a transition of an observer process)*
(c) $u \mapsto t, u' \mapsto t' \in \varphi$ *(subject and observer transitions are linked via the observation connection)*

One could say that an observer is a historian with a preconceived idea about subject process behaviour. An observer would not tolerate a subject that does not follow a certain plan or timetable. Note the use of $\varphi \subseteq \alpha\mathsf{S} \times \alpha\mathsf{C}$ to define the alphabet of a timed system interpretation. Whenever we speak about a timed system we always imply, unless specifically indicated otherwise, that the timed system permits an interpretation.

It is essential to note that (despite the nomenclature) the observer is an integral part of the timed system, and does not have a merely passive role. The observer characterises the timing constraints that the developer wishes to impose on an otherwise untimed system, and permits only interpretations that conform to these constraints.

Recipe 2 (Consistency) It may happen that a proof of liveness and timing properties is merely a consequence of an incompatibility between the observer and the subject process. This incompatibility results in a vacuous interpretation of a timed system that defines no common state transitions. To avoid this problem, it is sufficient to exhibit an initialisation of the timed system. For a timed system $\mathsf{C} \cdot \varphi \cdot \mathsf{S}$ one needs to prove that

$$\exists x, y \cdot x \mapsto y \in \iota\mathsf{S} \times \iota\mathsf{C} \wedge x \mapsto y \in \varphi \tag{1}$$

Condition 1 is called the *consistency proof obligation* of a timed system. □

The consistency condition holds for the system in Example 1; one possible witness is mapping $nil \mapsto nil$.

We give now the condition under which an event may be safely removed from a timed system without affecting the overall behaviour.

Recipe 3 (Relation empty) Consider a timed system $\mathsf{C} \cdot \varphi \cdot \mathsf{S}$ with Event-B machine S defining some event E_i:

$$E_i = \textbf{any } p_i \textbf{ where } G_i(p_i, v) \textbf{ then } S_i(p_i, v, v') \textbf{ end}$$

Let S' be a machine identical to S except that E_i is suppressed:

$$E_i = \textbf{any } p_i \textbf{ where } \bot \textbf{ then } S_i(p_i, v, v') \textbf{ end}$$

Timed systems $\mathsf{C} \cdot \varphi \cdot \mathsf{S}$ and $\mathsf{C} \cdot \varphi \cdot \mathsf{S}'$ are equivalent provided the following condition is satisfied

$$(\varphi[\mathsf{before}(E_i)] \times \varphi[\mathsf{after}(E_i)]) \cap c = \varnothing \tag{2}$$

where $\mathsf{before}(e)$ corresponds to the enabling states defined by an event guard and $\mathsf{after}(e)$ is a set of possible new states computed by an event:

$$\mathsf{before}(e) = \{v \mid I(v) \wedge \exists p_i \cdot G_i(p_i, v)\}$$
$$\mathsf{after}(e) = \{v' \mid I(v) \wedge \exists p_i \cdot (G_i(p_i, v) \wedge S_i(p_i, v, v'))\}$$

The technique allows one to prove that after removing event E_i the overall timed system does not become less live since the E_i is already prevented from occuring by an observer.

Example 2 (Buffer, contd.). We can apply the event removal technique to prove that timed system $\mathsf{C_0} \cdot \varphi \cdot \mathsf{BUF}$ from Example 1 does indeed define a lossless buffer.

To make the buffer lossless, we need to rule out the posibility of event *wr* writing into a non-empty buffer. That is, event *wr* should not happen when $b \neq nil$. Event *wr* may be represented (via a trivial case of refinement) by the following two events.

$$wr = \textbf{refines } wr \textbf{ any } v \textbf{ where } \mathbf{b = nil} \wedge v \in V1 \textbf{ then } b := v \textbf{ end}$$
$$owr = \textbf{refines } wr \textbf{ any } v \textbf{ where } \mathbf{b \neq nil} \wedge v \in V1 \textbf{ then } b := v \textbf{ end}$$

It is possible to prove that *owr* is not a part of the timed system $\mathsf{C_0} \cdot \varphi \cdot \mathsf{BUF}$ by showing that Condition 2 holds for *owr*:

$$(\varphi[\mathsf{before}(owr)] \times \varphi[\mathsf{after}(owr)]) \cap c = \varnothing$$

which expands to $\varphi[\{b \mid b \in V1 \wedge (\exists v \cdot v \in V1)\}] \times \varphi[\{b' \mid b \in V1 \wedge (\exists v \cdot v \in V1 \wedge b' = v)\}] \cap c = \varnothing$. Since $V1$ is not empty we have that $\exists v \cdot v \in V1 \Leftrightarrow \top$ and also $V1 = \{b \mid b \in V1\}$. The condition simplifies to $\varphi[V1] \times \varphi[V1] \cap c = \varnothing \Leftrightarrow \varphi[V1] \times \varphi[V1] \cap (V \times V \setminus (V1 \times V1)) = \varnothing \Leftrightarrow \top$. Hence, we can replace machine BUF in $\mathsf{C_0} \cdot \varphi \cdot \mathsf{BUF}$ with the following machine $\mathsf{BUF'}$:

machine BUF′
  ...
  events
    wr = **any** v **where** $\mathbf{b = nil} \wedge v \in V1$ **then** $b := v$ **end**
    rd = **begin** $b := nil$ **end**
end

It is trivial to see that $\mathsf{BUF'}$ defines a lossless buffer. Hence, $\mathsf{C_0} \cdot \varphi \cdot \mathsf{BUF}$ is also a lossless buffer. □

It is often advantageous to deal with an observer that is cooperative enough to completely accept any execution of a subject process. Then one knows a priori that something happens in a subject process for every possible point of time defined by an observer.

Definition 6 (Strictness). *A timed system* $\mathcal{A} = (\alpha\mathsf{C}, c, \iota\mathsf{C}) \cdot \varphi \cdot (\alpha\mathsf{S}, s, \iota\mathsf{S})$ *is strict if for every* $u \mapsto t \in \alpha\mathsf{S} \times \alpha\mathsf{C}$ *and* $t \mapsto t' \in c$ *there exists some* u' *such that* $(u \mapsto t) \mapsto (u' \mapsto t') \in \tau\mathcal{A}$ *and* $\iota\mathsf{C} \subseteq \varphi[\iota\mathsf{S}]$.

In a system with a strict observer, an observation connection is also a simulation relation [3].

Example 3 (Buffer, contd.). Observer C_0 permits a concise abstraction however there is an even simpler observer that achieves the same effect. Notice that $\mathsf{C_0} \cdot \varphi \cdot \mathsf{BUF}$ defines three transitions classes: reading a value and setting buffer to 0 ($V^+ \times \{nil\}$); reading an empty buffer ($\{nil \mapsto nil\}$); writing into an empty buffer ($\{nil\} \times V^+$). We shall exploit this property and define a new observer C_1 such that these three classes are the kernels of new observation connection φ_1:

context $\mathsf{C_1}$
  extends *def*
  sets $\alpha\mathsf{C_1}$
  constants $c_1, \varphi_1, \iota\mathsf{C_1}, \mathrm{E}, \mathrm{F}$
  axioms
    $\mathrm{partition}(\alpha\mathsf{C_1}, \{\mathrm{E}, \mathrm{F}\})$
    $\iota\mathsf{C_1} = \{\mathrm{E}, \mathrm{F}\}$
    $c_1 = \{\mathrm{E} \mapsto \mathrm{E}, \mathrm{E} \mapsto \mathrm{F}, \mathrm{F} \mapsto \mathrm{E}\}$
    $\varphi_1 = V1 \times \{\mathrm{F}\} \cup \{nil\} \times \{\mathrm{E}\}$
end

It is not hard to see that event removal condition also holds for $\mathsf{C_1} \cdot \varphi_1 \cdot \mathsf{BUF}$: $\varphi_1[\mathsf{before}(owr)] \times \varphi_1[\mathsf{after}(owr)] \cap c_1 = \varnothing \Leftrightarrow (\{\mathrm{F}\} \times \{\mathrm{F}\}) \cap c_1 = \varnothing \Leftrightarrow \top$. It is easy to see that, unlike $\mathsf{C_0} \cdot \varphi \cdot \mathsf{BUF}$, system $\mathsf{C_1} \cdot \varphi_1 \cdot \mathsf{BUF}$ is *strict*. □

The fourth recipe allows a developer to show that a state which is possible in the untimed process is ruled out by the timing constraints. We give the theory of the technique and demonstrate it with a simple example.

Recipe 4 (Point empty) Consider a timed system $\mathsf{C} \cdot \varphi \cdot \mathsf{S}$ and a subject state $w \in \alpha\mathsf{S}$. If one can show that φ does not project w into anything at all in $\alpha\mathsf{C}$ then, by the Definition 5 of timed system interpretation, any state $\chi \in \alpha\mathsf{C} \times \alpha\mathsf{S}$ where $\mathrm{prj}_2[\{\chi\}] = \{w\}$ is not a state of $\mathsf{C} \cdot \varphi \cdot \mathsf{S}$.

Thus, a subject state not projected by φ is not reachable in a timed system. A proof that assumes the existence of such a state may be discharged by deriving a contradiction with the following rule.

$$\forall W \cdot W \subseteq \Omega \wedge \varphi[W] = \varnothing \Rightarrow \bot \tag{3}$$

where $\Omega = \{v \mid I(v)\}$ is the set of subject states. □

Example 4 (Mutex). In this example we describe a very simple mutual exclusion algorithm that works due to a rigid scheduling of the involved threads. The state of a thread p is defined by $s(p)$ and is one of the following values: 'out', denoting

that p is outside of a critical section and not trying to enter it; 'prep', telling that the thread is about to enter the critical section; and 'in' for the states when the thread is in the critical section.

machine MTX
  variables s
  invariant
    inv1 : $s \in P \to \{\text{out}, \text{prep}, \text{in}\}$
    inv2 : $\text{card}(s^{-1}[\{\text{in}\}]) \leq 1$
  initialisation $s := P \times \{\text{out}\}$
  events
    prepare = **any** p **where** $p \in P \wedge s(p) = \text{out}$ **then** $s(p) := \text{prep}$ **end**
    enter = **any** p **where** $p \in P \wedge s(p) = \text{prep}$ **then** $s(p) := \text{in}$ **end**
    leave = **any** p **where** $p \in P \wedge s(p) = \text{in}$ **then** $s(p) := \text{out}$ **end**
end

where set P of processes is finite. Invariant inv2 expresses the property of mutual exclusion. We employ the following observer process to define that no two processes may be, at the same time, at stages 'prep' and 'in':

context C
  ...
  $c \subseteq \alpha\mathsf{C} \times \alpha\mathsf{C}$
  $S = \mathbb{P}(P \times \{\text{out}, \text{prep}, \text{in}\})$
  $\varphi \subseteq S \times \alpha\mathsf{C}$
  axm5 : $\forall t, q \cdot t, q \in P \wedge t \neq q \Rightarrow [\![s(t) = \text{prep} \wedge s(q) = \text{in}]\!] = \varnothing$
end

where $[\![P(\omega)]\!] \equiv \varphi[\{\omega \mid P(\omega)\}]$. The only non-trivial proof obligation in this model is the preservation of inv2 by event *enter*. It asks to prove, for some process p, that entering the critical does not violate safety invariant inv2.

$$\text{card}(s^{-1}[\{\text{in}\}]) \leq 1 \wedge s(p) = \text{prep} \vDash \text{card}((s \lhd\!\!\!\!- \{p \mapsto \text{in}\})^{-1}[\{\text{in}\}]) \leq 1$$

The condition cannot be discharged within the scope of the subject model alone. We need to bring in the constraints of the observer model to demonstrate the condition. We proceed by replacing $\text{card}((s \lhd\!\!\!\!- \{p \mapsto \text{in}\})^{-1}[\{\text{in}\}]) \leq 1$ with a stronger goal $s^{-1}[\{\text{in}\}] = \varnothing$ and continue with a proof by contradiction. The negation of $s^{-1}[\{\text{in}\}] = \varnothing$ in hypothesis gives

$$s(p) = \text{prep} \wedge s(x) = \text{in} \wedge x \neq p \vDash \bot$$

A state where one process is in the critical section and the other is about to enter the critical section is disallowed by the observer (axm5) so that the point empty technique may be used to discharge the condition. Instantiating axm5 with $t = p, q = x$ we have $\varphi[\{a \cdot a \in S \wedge a(p) = \text{prep} \wedge a(x) = \text{in} \mid a\}] = \varnothing$ which gives us set W to instantiate Condition 3 and derive a contradiction in hypothesis.

One way to realise observer C is by defining it to be cyclic scheduler that allows processes to access the critical section at fixed time intervals. □

4 Realisability

According to Definition 5, a timed system is a transition (or a process, as it is defined in Definition 2). Hence, a timed system may itself be employed in the role of subject or observer and one can define a complex timed system made of subsystems which are also timed systems. One application of the compositionality property is a structure called *time log*. A time log is timed system observed by an external observer. Informally, the external observer make a record of observation using its own timekeeping device.

Definition 7 (Time log). *Time log* $\mathbb{T}(\mathsf{C}\cdot\varphi\cdot\mathsf{S})_{\mathsf{T},\omega}$ *of timed system* $\mathcal{A} = \mathsf{C}\cdot\varphi\cdot\mathsf{S}$ *is a process*

$$\mathbb{T}(\mathcal{A})_{\mathsf{T},\omega} = (\varphi; \omega, L[\tau(\mathsf{T}\cdot\omega\cdot\tau(\mathcal{A}))], \iota\mathsf{T})$$

where $\mathsf{T}\cdot\omega\cdot\tau(\mathcal{A})$ *is strict,* ω *is total and functional; projection* L *removes states of process* C*:* $L[X] = \{((a,b),\{c\}) \mapsto (a,c) \mid ((a,b),\{c\}) \in X\}$*. Also,* $L[\iota\mathcal{A}\times\iota\mathsf{T}]\cap\omega \neq \varnothing$*.*

A time log defines a timed system (or a process) which does not reference states of observer C. A time log is itself a transition system hence it is sometimes possible to replace a timed system with its time log. One common reason to do this is to separate the proof of logical properties relevant to timing constraints from the proof of how these properties may be expressed in a specific, implementation-oriented form, e.g., hard real-time constraints.

Not all timed system may be realised in physical reality. If the object of a formal development is a piece of software of hardware it is necessary to check, at the level of a concrete design, that certain properties are respected by an observer process. These properties, called realisability conditions, are as follows:

- time advance is monotonic
- infinite subject activities take infinitely long time to observe

Instead of checking these properties directly on an observer model, it is more convenient to consider yet another observer and study the observations defined by the new observer and the original timed system. The first realisability condition demands that the time model described in an observer process may be mapped to a monotonic time model. The second condition prohibits a situation where an unterminating activity of a subject process is timed to terminate by a certain deadline.

An animation is a time log satisfying the the realisability conditions. Let bounded(X,Y) denote the fact that there exist lower and upper bounds w.r.t. the relation of process $\mathsf{Y} = (\alpha\mathsf{Y}, <)$, bounded$(X) \equiv X \subseteq \alpha\mathsf{Y} \wedge (\exists l, u \cdot l, u \in \alpha\mathsf{Y} \wedge \forall p \cdot p \in P \Rightarrow l < p < b)$.

Definition 8 (Animation of timed system). *An animation of* $\mathsf{C} \cdot \varphi \cdot \mathsf{S}$ *is a time log* $\mathbb{T}(\mathsf{C} \cdot \varphi \cdot \mathsf{S})_{\mathsf{T},\omega}$ *such that* T *is monotone and every bounded subset of* $\alpha\mathsf{T}$ *maps to a finite sequence of subject actions:* $\forall P \cdot \text{bounded}(P, \mathsf{T}) \Rightarrow \text{finite}(\omega^{-1}; \varphi^{-1}[P])$.

Definition 9 (Realisability). *A timed system is realisable if it admits at least one animation.*

According to Definition 5, a timed system is a transition (or a process, as it is defined in Definition 2). Hence, a timed system may itself be employed in the role of subject or observer and one can define a complex timed system made of subsystems which are also timed systems. One application of the compositionality property is a structure called an *animation*. An animation is a timed system formed by observed another timed system. Informally, the external observer make a record of observation using its own timekeeping device.

Definition 10 (Animation). *Animation* $\mathbb{T}(\mathsf{C} \cdot \varphi \cdot \mathsf{S})_{\mathsf{T},\omega}$ *of timed system* $\mathcal{A} = \mathsf{C} \cdot \varphi \cdot \mathsf{S}$ *is a process*

$$\mathbb{T}(\mathcal{A})_{\mathsf{T},\omega} = (\varphi; \omega, L[\tau(\mathsf{T} \cdot \omega \cdot \tau(\mathcal{A}))], \iota\mathsf{T})$$

where $\mathsf{T} \cdot \omega \cdot \tau(\mathcal{A})$ *is strict,* ω *is total and functional; projection* L *removes states of process* C*:* $L[X] = \{((a,b),\{c\}) \mapsto (a,c) \mid ((a,b),\{c\}) \in X\}$. *Also,* $L[\iota\mathcal{A} \times \iota\mathsf{T}] \cap \omega \neq \varnothing$.

An animation defines a timed system (or a process) which does not reference states of observer C. An animation is itself a transition system hence it is sometimes possible to replace a timed system with its animation. One common reason to do this is to separate the proof of logical properties relevant to timing constraints from the proof of how these properties may be expressed in a specific, implementation-oriented form, e.g., hard real-time constraints.

Recipe 5 (Real-time constraints) The form of timing constraints of a concrete design is dictated by the practical necessity to validate constraints via some form of static analysis (i.e., worst-case execution time) or by observing execution runs of a software or hardware implementation. For the former, it may be necessary to present constraints in the form of durations of elementary execution steps.

In a timed Event-B we suggest to use an observer based on a dense linear order (DLO) to realise real-time constraints. A linear order c is dense if it satisfies condition $\forall x, y \cdot x \mapsto y \in c \Rightarrow (\exists z \cdot x \mapsto z \in c \wedge z \mapsto y \in c)$. Intuitively, with a dense order one is able to define durations and time points with an arbitrary precision[1]. It also means one is able to interpolate between any two time points. An example of an Event-B model of a DLO may be found in [1].

Assume a process $\mathsf{C} = (\alpha\mathsf{C}, c, \iota\mathsf{C})$ where c is a DLO. This process will be employed to define an animator for some timed system $\mathcal{A}$. We introduce a layer

[1] This is what, we believe, is usually meant as a property distinguishing a "real-valued clock" from a "discrete" clock.

of syntactic shorthand to express properties of C. Let $t \in \alpha\mathsf{C}$ be the current state of animator C, P a predicate defined on set $\gamma^{-1}[\alpha\mathsf{C}]$ and γ an animation relation.

- $\mathsf{at}(P,t)_\gamma \equiv P(\gamma^{-1}(t))$: event (defined by predicate) P happens at time t;
- $\mathsf{during}(P,i)_\gamma \equiv \exists t \cdot t \in i \Rightarrow \mathsf{at}(\mathrm{s},t)_\gamma$: event P happens at least once during time interval i;
- $\mathsf{within}(\Delta,P,t)_\gamma \equiv \exists t' \cdot t' > t \wedge t' - t \leq \Delta \wedge P(\gamma^{-1}(t'))$: event P happens within Δ time units after t;
- $\mathsf{after}(\Delta,P,t)_\gamma \equiv \neg\mathsf{within}(\Delta,P,t)_\gamma \wedge \mathsf{within}(\infty,P,t)_\gamma$: event P happens not sooner than Δ time units (but happens eventually) after t.

The following statement says that, at all times, whenever there happens a stimulus event ($\mathbf{s}$), it is followed by a response event ($\mathbf{r}$) within Δ time units

$$\forall t \cdot t \in \alpha\mathsf{C} \wedge \mathsf{at}(\mathbf{s},t)_\gamma \Rightarrow \mathsf{within}(\Delta,\mathbf{r},t)_\gamma$$

One may elect to be more precise and state that, should a stimulus happen at time t, a response follows within interval $[t+\Delta_1, t+\Delta_2]$

$$\forall t \cdot t \in \alpha\mathsf{C} \wedge \mathsf{at}(\mathbf{s},t)_\gamma \Rightarrow \mathsf{during}([t+\Delta_1, t+\Delta_2],\mathbf{r})_\gamma$$

which is the same as

$$\forall t \cdot t \in \alpha\mathsf{C} \wedge \mathsf{at}(\mathbf{s},t)_\gamma \Rightarrow \neg\mathsf{within}(\Delta_1,\mathbf{r},t)_\gamma \wedge \mathsf{within}(\Delta_2,\mathbf{r},t)_\gamma$$

As a further illustration, the following are statements about a simple traffic light.

- $A_\gamma(t) \equiv \mathsf{within}(\infty,\mathbf{green},t)_\gamma$: the green aspect is eventually lit;
- $B_\gamma(t) \equiv \neg\mathsf{within}(\infty,\mathbf{green}\wedge\mathbf{red},t)_\gamma$: the green and red aspects are never lit at the same time;
- $C_\gamma(t) \equiv \mathsf{at}(\mathbf{red},t) \Rightarrow \mathsf{after}(\Delta_1,\mathbf{green},t)_\gamma$: the green aspect follows the red aspect within Δ_1 time units;
- $D_\gamma(t) \equiv \mathsf{at}(\mathbf{red},t) \Rightarrow \neg\mathsf{after}(\Delta_2,\mathbf{yellow},t)_\gamma$: the yellow aspect might be lit after red aspect might happen but not sooner than Δ_2 time units.

Any traffic light implementation must respect these properties. In an animator specification this is expressed by addign an axiom that animator C may never violate these properties:

$$\iota\mathsf{C} \subseteq \mathbf{v}_\gamma \qquad c[\mathbf{v}_\gamma] \subseteq \mathbf{v}_\gamma \tag{4}$$

For the traffic light example, it must hold that the initial states $\iota\mathsf{C}$ satisfies properties $A-D$ and the animation relation c preserves properties $A-D$ so that $\mathbf{v}_\gamma$ is the set of all valid time points: $\mathbf{v}_\gamma = \{t \mid t \in \alpha\mathsf{C} \wedge A_\gamma(t) \wedge B_\gamma(t) \wedge C_\gamma(t) \wedge D_\gamma(t)\}$.

The verification effort is in showing that an animator does indeed animate a given timed system. For this we consider the animation relation γ connecting the timed system observer with the animator C. If one can prove that γ exists as an

animation relation that one may take the animation as a timed specification that respects both the abstract scheduling properties of the original timed system and the real-time constraints of the animator. □

Example 5 (Buffer, contd.). In this example we show how to construct an animation of the lossless buffer timed system $\mathsf{C_1} \cdot \varphi_1 \cdot \mathsf{BUF}$ in the terms of the relative speeds of the write and read operations. For this, we reinterpret the timing requirements with an animator that explicitly defines operation delays and time-outs.

Consider a DLO animator $\mathsf{T} = (\alpha\mathsf{T}, <, \{zero\})$ and animation relation $\omega \in \{\mathrm{E}, \mathrm{F}\} \to \alpha\mathsf{T}$ such that for all $x, y \in \alpha\mathsf{T}$ it holds that

(a) $P_1(t) \equiv \mathsf{within}(\Delta_R, \lambda c \cdot c = \mathrm{E}, t)_\omega$ (a read happens within Δ_R time units)
(b) $P_2(t) \equiv \neg\mathsf{within}(\Delta_W, \lambda c \cdot c = \mathrm{F}, t)_\omega$ (a write happens not sooner than Δ_W time units from now)
(c) $P_3 \equiv \Delta_R \leq \Delta_W$ (reader is quicker than writer)
(d) $\omega[\{\mathrm{E}\}] = \{zero\}$ (system starts at time $zero \in \alpha\mathsf{T}$)

In this example, properties $P_1 - P_3$ also sufficiently constrain the animation relation γ. Delays Δ_R and Δ_W define the durations of **wr** and **rd**. We prove that T animates $\mathsf{C_1} \cdot \varphi_1 \cdot \mathsf{BUF}$ with animation connection ω. Let $\mathbf{v}_\gamma = \{t \mid P_1(t) \wedge P_2(t)\}$ (we omit P_3 as it is not time-sensetive) and *less* be a prefix form of $<$. From Condition 4 we derive the following hypothesis

$$\{zero\} \subseteq \mathbf{v}_\omega \qquad less[\mathbf{v}_\omega] \subseteq \mathbf{v}_\omega$$

The initialisation condition expands to

$$\begin{array}{l}(\exists t' \cdot t' > zero \wedge t' - zero \leq \Delta_R \wedge \omega^{-1}(t') = \mathrm{E}) \wedge \\ \neg(\exists t' \cdot t' > zero \wedge t' - zero \leq \Delta_W \wedge \omega^{-1}(t') = \mathrm{F})\end{array} \tag{5}$$

Statement $less[\mathbf{v}_\omega] \subseteq \mathbf{v}_\omega$ gives

$$\begin{array}{l}\forall t_0, t_1 \cdot t_0 t_1 \in \alpha\mathsf{T} \wedge t_0 < t_1 \wedge \\ ((\exists t' \cdot t' > t_0 \wedge t' - t_0 \leq \Delta_R \wedge \omega^{-1}(t') = \mathrm{E}) \wedge \\ \neg(\exists t' \cdot t' > t_0 \wedge t' - t_0 \leq \Delta_W \wedge \omega^{-1}(t') = \mathrm{F})) \Rightarrow \\ \quad ((\exists t' \cdot t' > t_1 \wedge t' - t_1 \leq \Delta_R \wedge \omega^{-1}(t') = \mathrm{E}) \wedge \\ \quad\quad \neg(\exists t' \cdot t' > t_1 \wedge t' - t_1 \leq \Delta_W \wedge \omega^{-1}(t') = \mathrm{F}))\end{array} \tag{6}$$

We sketch the proof for the fact that ω is an animation relation: for any $t < t'$ it holds that $\omega^{-1}(t) < \omega^{-1}(t')$. The only case when $\omega^{-1}(t) \geq \omega^{-1}(t')$ is $\omega^{-1}(t) = \mathrm{F}$ and $\omega^{-1}(t') = \mathrm{F}$. Assume that such t and t' exist. From Condition 5 there exists $t_0 < t$ and $\omega^{-1}(t_0) = \mathrm{E}$. From Condition 6 we have the existence of t_1 such that $t_1 > t_0 \wedge t_1 - t_0 \leq \Delta_R \wedge \omega^{-1}(t_1) = \mathrm{E}$ and $t_1 > t'$. The $t_1 > t'$ part may be shown by induction: since t_0 and t are a finite distance apart there exists $t_i, \omega^{-1}(t_i) = \mathrm{E}$ such that there does not exist $t_{i+1}, \omega^{-1}(t_i) = \mathrm{E}$ and $t_{i+1} < t$. Also, from Condition 6, we have that $t' - t \geq \Delta_W$ which leads to a contradiction due to condition (c). □

Recipe 6 (Point merge) This technique is a generalisation of the empty point technique. It is used to derive a contradiction when a subject state, defined by the intersection of states of two or more concurrent threads disagrees with the observation model. The following lemma states how to make a transition from a set of statements about individual thread states to a statement about a time point when such a state configuration may be observed.

Lemma 1 (Point merge). *Let $\mathcal{W}$ and $\mathcal{P}_i$ be non-empty subject process states such that $\mathcal{W} = \{v \mid W(v)\}$, $\mathcal{P}_i = \{v \mid P_i(v)\}$ where $W(v)$ and $P_i(v)$ are predicates over subject process state space and it holds that $W \Rightarrow \bigwedge_i P_i$. Then there exist time points $t_i \in [\![P_i]\!] \cap [\![W]\!]$ such that $\forall i, j \cdot t_i = t_j$.*

Proof. See [3] (a Rodin Toolkit proof).

The proof technique is to show that no two states from P_i and P_j, $i \neq j$ may be observed at the same time (due to some timing conditions). Then the existence of a time point common for the two states P_i and P_j gives a contradiction.

We have applied the point merge technique in the proof of Fischer's timing-based algorithm of mutual exclusion [19, 4]. The complete Event-B development of the algorithm is available at [2]. □

5 Related Work

One closely related work is that of Abadi and Lamport [4] which shows that timing constraints may be expressed directly in TLA without syntactic or semantic extensions. Timed automata [6] offers a formal framework for specifying real-time properties by enriching the state of an automata with a number real-valued clocks. The UPPAAL[7] tool offers support for automated verification of timed automata. Timed process algebras have been researched extensively and there is a large variety of notations and semantics. The timed extensions of CSP [19] and CCS[16] are two notable examples.

Although Event-B lacks any native support of time, some form of timed modelling may be done directly in the Event-B notation. The basic principle - a clock variable employed to keep track of time - is fairly intuitive and has been applied in various state-based and proces algebraic methods. One example is Tock-CSP [19] which uses the standard CSP notation and measures the passage of time by counting the occurence of a *tock* event. A state-based equivalent is having a dedicated variable *now* to track the passage of time and express timing conditions [4, 15].

Previous work on modelling time in B uses a clock variable which records the current value of a clock, and an operation is given to advance time [9]. This approach is taken up again for Event B in [10, 17]. In [14] the concept of time is embedded into the B notation itself. Time is modelled by equipping a machine with a clock and assuming that an event execution is not instanteneous.

We briefly discuss the general ideas behind a clock variable technique and show how it may related to our approach. A machine with a clock variable t has the following form

machine m
  variables v, t
  invariant $I(v) \wedge P(v,t)$
  initialisation $R(v', t')$
  events
    sys = **when** $G(v,t)$ **then** $S(v,t,v')$ **end**
    $tick$ = **when** $H(v,t)$ **then** $T(t,v,t')$ **end**
end

Timing constraints are encoded as a safety invariant $P(v,t)$. A clock variable t is usually defined as $t \in \mathbb{N}$ to imply an unbounded discrete clock. The clock variable is updated by event $tick$; an update would either increment t by one or 'jump' time to some interesting point in future (i.e., next deadline). The behaviour of a system is then cumulatively defined by some event sys which may not update t but may refer to t in its guard. An informal interpretation is the following: if activity sys must happen within interval $[a(v), b(v)]$, guard $G(v,t)$ should not allow sys happen before $a(v)$ while the clock guard $H(v,t)$ should prevent time from progressing beyond $b(v)$ until sys has happened. Sometimes intervals are singular and one speaks about deadlines [17].

Verification conditions for time properties are invariant preservation theorems for $P(v,t)$:

$$I(v) \wedge P(v,t) \wedge G(v,t) \wedge S(v,t,v') \Rightarrow P(v',t)$$
$$I(v) \wedge P(v,t) \wedge H(v,t) \wedge T(t,v,t') \Rightarrow P(v,t')$$

On the left-hand side, $P(v,t)$ is stated on an old state and on the right-hand side on a new state produceed by respective events. What such theorems show is that, if properly initialised, the system is guaranteed to stay within the bounds set by predicate $P(v,t)$.

This technique allows one to demonstarte a range of progress properties with a heavy reliance on Event-B refinement principles. In Event-B one is able to refine a previously atomic transition into a sequence or a terminating loop of new transitions. Atomicity refinement - as this technique is known - allows one to prove certain progress properties by constructing suitable refinement relations. For instance, one can prove that activity A completes before activity B (each comprising several events) by showing that A and B are derived from abstract events a and b and, at that abstraction level, it is somehow known that a always precedes b. Such kind of properties of a and b may be demonstrated by encoding an ordering relation with an auxiliary variable or generating a special proof obligation [12, 13]. A model with a clock variable restates the atomicity refinement technique in terms of deadlines and intervals.

It is easy to convert the tick model into Leibnizian time. Rather than giving a timed system that defines an equivalent transitions system we define a timed system that, as we believe, corresponds to the intended purpose of an Event-B

machine with a *tick* event. Let $\Upsilon = \{t \mid \exists v \cdot P(v,t)\}$ and $\Omega = \{v \mid \exists t \cdot I(v) \wedge P(v,t)\}$. Then machine m corresponds to a timed system $\mathsf{C} \cdot \varphi \cdot \mathsf{S}$ such that

- $\mathsf{C} = (\Upsilon, c, \iota\mathsf{C})$ where $c \subseteq \Upsilon \times \Upsilon$ and $c \cap \mathrm{id}(\Upsilon) = \varnothing$
- $\mathsf{S} = (\Omega, s, \iota\mathsf{S})$ where $s = \{v \mapsto v' \mid \exists t \cdot G(v,t) \wedge S(v,t,v')\}$
- $\varphi = \{v \mapsto t \mid G(v,t) \wedge \neg H(v,t)\} \cap \{(v,t) \mid P(v,t)\}$

Note that $\mathsf{C} \cdot \varphi \cdot \mathsf{S}$ does not depend on the definition of $T(t,v,t')$. This is because in an Event-B model there is no meaning to $T(t,v,t')$ in the sense that no proof obligation constraints $T(t,v,t')$ beyond requiring that $T(t,v,t')$ is safe and irreflexive (should be imposed by the proof of convergence of *tick*) and $T(t,v,t') \subseteq \{(v,t) \mid P(v,t)\} \times \{(v,t) \mid P(v,t)\}$. The guard $H(v,t)$ of *tick* potentially matters as it would be a part of the deadlock freeness condition. In practice, the convergence and deadlock freeness of *tick* are hard to prove and are rarely attempted. In the Leibnizian time model the $P(v,t)$ constraint is placed in the observation connection and the irreflexivity property is a part of the observer model leaving subject S to contend with functional properties.

6 Discussion

We have presented a summary of our ideas on how the Leibnizian model of time may be used to construct timed Event-B specifications. Our approach offers a homogenous technique to time modelling where properties of timed models are expressed and proven in a gradual, refinement-based manner. The approach is a conservative extension of Event-B. No notational or semantical changes are necessary and the existing modelling tools have proven adequate.

Our technique does not dictate any specific time domain: we let a modeller choose the most appropriate abstraction of time – a simple scheduler, a fictious integer clock or a dense time clock. Both dense and discrete time domains are supported so that the approach may be used as a part of a toolchain with a wide range of potential roles including expressing scheduling properties and hard real-time constraints. The approach has proven to be quite effecient and intuitive: we were able to tackle several large case studies and, as far as we are aware, our models are simpler and require a lower verification effort while all proofs are completely machine-checked.

Due to space constraints, we did not present a larger case study although one such case study is available at [2]. Many recipes were not discussed. These include rules for demonstrating the realisability of a timed specification and several refinement-related recipes. We plan to provide a plug-in to the Rodin Toolkit [18] for automated generation of the timed systems proof obligations and a template-based assistant for constructing various kinds of observer processes.

References

1. A. Iliasov and J. Bryans. Dense linear order; An Event-B context encoding. online at http://www.iliasov.org/fischer/observer.pdf.

2. A. Iliasov and J. Bryans. Event-B development of Fischer's algorithm. online at http://iliasov.org/fischer.
3. A. Iliasov and J. Bryans. Supplementary material: proofs and models. online at http://iliasov.org/ltime.
4. M. Abadi and L. Lamport. An old-fashioned recipe for real time. In *Proceedings of the Real-Time: Theory in Practice, REX Workshop*, London, UK, UK, 1992. Springer-Verlag.
5. J.-R. Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010.
6. R. Alur and D. L. Dill. Automata for modeling real-time systems. In *Proceedings of the seventeenth international colloquium on Automata, languages and programming*. Springer-Verlag New York, Inc., 1990.
7. J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, and W. Yi. Uppaal - a tool suite for automatic verification of real-time systems. In *Proceedings of the DIMACS/SYCON workshop on Hybrid systems III : verification and control*, pages 232–243. Springer-Verlag New York, Inc., 1996.
8. J. W. Bryans, J. S. Fitzgerald, A. Romanovsky, and A. Roth. Patterns for modelling time and consistency in business information systems. In *15th IEEE International Conference on Engineering of Complex Computer Systems*. IEEE Computer Society, 2010.
9. M. Butler and J. Falampin. An approach to modelling and refining timing properties in B. In *Refinement of Critical Systems (RCS)*, January 2002.
10. D. Cansell, D. Méry, and J. Rehm. Time Constraint Patterns for Event B Development. In *Formal Specification and Development in B, 7th International Conference of B Users*, 2007.
11. M. J. Futch. *Leibniz's Metaphysics of Time and Space*. Springer-Verlag GmbH, 2008.
12. Thai Son Hoang and Jean-Raymond Abrial. Reasoning about liveness properties in event-b. In *ICFEM*, pages 456–471, 2011.
13. A. Iliasov. Use case scenarios as verification conditions: event-b/flow approach. In *Proceedings of the Third international conference on Software engineering for resilient systems*, SERENE'11, 2011.
14. K. Lano. *The B Language and Method: A Guide to Practical Formal Development*. Springer-Verlag New York, Inc., 1996.
15. Nancy Lynch and Frits Vaandrager. Forward and backward simulations - part ii: Timing-based systems. *Information and Computation*, 128.
16. F. Moller and C. Tofts. A temporal calculus of communicating systems. In *Proceedings on Theories of concurrency : unification and extension: unification and extension*, CONCUR '90, pages 401–415. Springer-Verlag New York, Inc., 1990.
17. J. Rehm. A method to refine time constraints in event B framework. In *AVoCS*, 2006.
18. RODIN. Event-B Platform. http://www.event-b.org/, 2009.
19. S. Schneider. *Concurrent and Real Time Systems: The CSP Approach*. John Wiley & Sons, Inc., New York, NY, USA, 1999.