Refinement of Timing Constraints for Concurrent Tasks with Scheduling

Chenyang Zhu, Michael Butler and Corina Cirstea

School of Electronics and Computer Science
University of Southampton
Southampton, United Kingdom
{cz4g16,mjb,cc2}@ecs.soton.ac.uk

Abstract. Event-B is a refinement-based formal method that is used for system-level modeling and analysis of concurrent and distributed systems. Work has been done to extend Event-B with discrete time constraints. However the previous work does not capture the communication and competition between concurrent processes. In this paper, we distinguish task-based timing properties with scheduler-based timing properties from the perspective of different system design phases. To refine task-based timing properties with scheduler-based timing properties based on existing trigger-response patterns, we introduce a nondeterministic queue based scheduling framework to schedule processes under concurrent circumstances, which addresses the problems of refining deadline constraint under concurrent situations. Additional gluing invariants are provided to this refinement. To demonstrate the usability of the framework, we provide approaches to refine this framework with FIFO scheduling policy as well as deferrable priority based scheduling policy with aging technique. We demonstrate our framework and refinement with a timed mutual exclusion case study. The model is proved using the Rodin tool.

Keywords: Event-B, Refinement, Timing, Concurrency, Scheduling

1 Introduction

Cyber Physical Systems (CPS) have received much attention in recent years due to their capabilities with advanced processors, sensors and wireless communication. Timing and concurrency are two key features of CPS [4]. The physical world evolves with time and this needs to be taken into account of within the computing devices in CPS. Real-time constraints need to be introduced to computing devices to ensure that the devices are interacting with the physical world correctly. What's more, with the advanced processors of CPS, multiple threads of computation are executing concurrently to achieve the goal of computation as a whole. For this reason understanding models and design principles for timing and concurrency is critical for CPS. In addition, it is difficult to model a complicated CPS with all the detailed features in one step. An abstraction and refinement approach can be adapted to manage complexity by modeling the system from abstract level to more concrete levels with reasoning to verify the consistency between refinement levels [7].

Event-B is a formal method for system-level modeling and analysis that is based on predicate logic and set theory [2]. Apart from its ability to model systems with precise mathematical abstractions, it also provides notions of abstraction and refinement. However, an explicit notion of real-time is not supported in Event-B, while real-time performance is critical for CPS. Existing work that extends Event-B models with timing properties uses a trigger-response pattern to model discrete time [16]. The pattern sets timestamps for trigger and response events and uses a tick event to prevent the global clock proceeding to a point where time constraints between trigger and response events would be violated. This pattern, however, does not distinguish timing properties for different system design phases. We define task-based timing properties as high level timing properties from system requirement specification phase, which place discrete time constraint on individual processes or tasks. These task-based timing properties can not describe the concurrent behaviour of tasks precisely. In real time systems, there are always several tasks running concurrently. High level time constraints for each task cannot guarantee the timing behaviour of the whole system. To model the behaviour of these concurrent processes, we define scheduler-based timing properties as concrete timing properties for the system design phase, which place discrete time constrains on the scheduler which schedules the concurrent tasks. Fairness can be imposed to restrict the nondeterministic behaviour of concurrent tasks. In many real-time applications, the weak guarantee of eventual occurrence of some event with weak/strong fairness assumption may be insufficient [11]. Alur and Henzinger proposed the definition of finitary fairness to impose a bound on the relative frequency in scheduling a set of events [3]. This definition of weak fairness requires that there is an unknown bound k for every computation of a system such that no enabled transition is postponed more than k consecutive steps.

We propose a nondeterministic queue based scheduling framework based on the idea of finitary fairness. Processes are placed in a nondeterministic position in the queue and once a process enters the queue, it cannot be postponed for more than k consecutive times. Additional gluing invariants are provided to use the framework to refine task-based deadline constraints with scheduler-based deadline constraints. Our approach is demonstrated by a timed mutual exclusion case study. Two alternative refinements from the nondeterministic queue to a FIFO scheduling policy as well as a deferrable priority based scheduling policy with aging technique are used to demonstrate the usability of the framework. In addition, we only need to prove the timing is satisfied by the nondeterministic queue since any refinement of the nondeterministic queue wll also satisfy the same timing deadlines.

Section 2 introduces some related work on modeling discrete time in Event-B with trigger-response patterns. We also discuss some additional fairness assumptions on the tick event introduced in the trigger-response pattern. In Section 3 we introduce task-based deadlines used in the requirement specification phase.

We use a timed mutual exclusion case study to illustrate the usage of task-based timing property. Section 4 refines the task-based deadlines to scheduler-based deadlines with a nondeterministic queue based scheduling policy as well as some additional gluing invariants. Section 5 gives two different implementations of the nondeterministic queue based scheduling policy. Section 6 gives the conclusion and some future work.

2 Related Work

2.1 Event-B

Event-B [2] is a formal modeling method based on set theory and first-order logic, which is usually used for system-level modeling and analysis with abstraction, refinement and reasoning on the model. Formal modeling is used to address the problem of lack of precision of specifications. However, formality on its own does not handle the problem of complex requirements and specifications [7]. Refinement helps to simplify the process of modeling with a stepwise approach. Gluing invariants which refer to variables of abstract and concrete machines are used to relate the states of concrete and abstract machines during refinement steps [14].

2.2 Time modeling

Timing issues are critical in cyber physical systems. Timing analysis should be carried out together with the development of the system to improve the real-time performance as well as guarantee the safety of the whole system. Timed automata [5] that are supported by the UPPAAL [15] model checker have been used in industrial modeling of real time systems. It is challenging to model a complex system with the timed automata formalism and UPPAAL as it does not support refinement of the model. Some approaches such as counter example guided abstraction refinement have been brought up to add abstraction and refinement when modeling a complex system [12]. This approach uses a model checker to get the counter examples from the abstract model and uses these counter examples as guides to refine the system. However it is difficult to find the missing part from the model just from counter examples.

Event-B is a general purpose modeling language that lacks explicit support for expressing and verifying timing constraints [19]. Work has been done to add time constraints to Event-B. Butler and Falampin proposed an approach to model and refine timing properties in classical B [1], which adds a clock variable representing the current time and an operation which advances the clock [9]. The approach ensures the timing properties are satisfied by preventing behaviours in which the clock advances to a point where deadlines would be violated. More work concerning time constraints such as delay, expiry, deadline and interval are presented recently [10, 16, 19]. These approaches define timing properties between different events, while Graf and Prinz introduce time to state transition systems [13].

2.3 Trigger-response pattern

To formally model the timing properties for the trigger-response pattern in control systems, Sarshogh and Butler proposed an approach that categorizes timing constraints in three groups: delay, expiry and deadline [16], which are denoted in (1a), (1b), (1c) below. All these three timing constraints follows a trigger-response pattern where trigger and response events are modeled as events in Event-B. (1a) shows that the *Response* event can only happen if the *delay* period has passed following the occurrence of the $Trigger$ event. (1b) shows that if the *expiry* period has passed then the *Response* event can never happen. (1c) denotes that if the $Trigger$ event occurs, then one of the events $Response_1$ to $Response_n$ must occur before *deadline* passes. To model these three timing properties in Event-B, a global clock as well as tick events are added to model the discrete time.

$$Delay(Trigger, Response, delay) \tag{1a}$$
$$Expiry(Trigger, Response, expiry) \tag{1b}$$
$$Deadline(Trigger, Response_1 \vee .. \vee Response_n, deadline) \tag{1c}$$

Figure 1 shows the trigger response pattern as an Event-B machine for the delay and deadline constraints, where *trigger* and *response* are modelled as events. The response event *response* must occur within time *ddl* of trigger event *trigger* occurring and can only occur if the delay period has passed. We use tT to refer to the time that trigger event happens, and we use tR to refer to the time that response event happens. Invariant @$inv1$ and @$inv2$ specify the delay and deadline timing property between *trigger* and *response* respectively. Guard @$grd3$ of the *response* event guarantees that the response is disabled when the global clock has not passed the delay period thus preserving @$inv1$. Guard @$grd1$ of the $Tick$ event constrains the global clock not to tick when the response event is missing its deadline thus preserving @$inv2$. @$inv3$ is needed to prove invariant @$inv2$. When modeling the expiry time constraint, @$grd3$ of *response* event should be $clk < tT + expiry$ to guarantee that the response is disabled when the global clock has passed the expiry period.

There are several patterns developed by Sarshogh to refine deadlines, delay and expiry. For example, to refine an abstract deadline D to sequential sub-deadlines $D_1..D_n$, there should be invariants to ensure the order of sequential sub-deadlines and the sum of the duration of sub-deadlines should be less than the abstract deadline duration [16].

Sarshogh's approach only handles the system with trigger and response pattern without specifying some possible interrupt events from the environment. To model a more complex system that supports interrupt events to interrupt current time intervals, Sulskus et al brought up the notion of time interval constraints in Event-B [19].

The trigger-response pattern only models discrete time constraints, while the real-world events do not always happen at integer-value times. Continuous time can be modelled approximated by choosing the granularity of the global clock,

```
invariants
    @inv1 tT<tR⇒tR−tT≥ dly
    @inv2 tR≥ tT⇒tR−tT≤ ddl
    @inv3 t=TRUE∧r=FALSE⇒clk−
        tT≤ ddl
  event trigger
    where
      @grd1 t=FALSE
      Ga(c,v)
    then
      @act1 t:= TRUE
      @act2 tT:= clk
      Act_a
  end
```

```
event response
  where
    @grd1 t=TRUE
    @grd2 r=FALSE
    @grd3 clk≥ tT+dly
    Gb(c,v)
  then
    @act1 r:= TRUE
    @act2 tR:= clk
    Act_b
end

event Tick
  where
    @grd1 t=TRUE∧r=FALSE⇒clk
        +1−tT≤ ddl
  then
    t:=  FALSE
    r:=  FALSE
end
```

Fig. 1: Model Timing Properties of Trigger-Response Events with Delay and Deadline

which model the timed system with an approximate sense. Banach et al presents the Hybrid Event-B extension which accommodates continuous behaviours in between discrete transitions [6]. Based on this extension, Butler et al outlines an approach to modeling and reasoning about hybrid systems which uses continuous functions over real intervals to model the evolution of continuous values over time [8].

The trigger-response pattern also introduces a *Tick* event to proceed the global clock without any fairness assumption. Without fairness, a valid event trace may repeat trigger and response events without executing any *Tick* event, which makes the global clock never proceed. Guard @$grd1$ of *Tick* event from Figure 1 shows that the only event that disables *Tick* event is itself. With weak fairness assumption on the *Tick* event, the *Tick* event is guaranteed to proceed the global clock in the system if the *Tick* event is enabled.

3 Task-Based Deadline Constraint

During the system design level, requirement specification are used to specify some high level specifications. We define task-based timing properties as high level timing properties to specify time constraints of individual tasks or processes. To better explain the definition, we use a simple timed mutual exclusion case study to demonstrate the usage of the framework. The timed mutual exclusion case study has two minimum requirements:

- No more than one process can be in its critical section at any time.
- If a process wishes to enter its critical section, it will enter the critical section within a certain deadline.

In the most abstract level, a mutual exclusion model is proposed which guarantees no two processes can be in the critical section at the same time. However a process can enter the critical section multiple times without allowing other process to proceed. Figure 2 gives the abstract mutual exclusion model. Here we use quantified variable p to represent one process. The event $wish(p)$ models the point at which process p wishes to enter the critical section. Event $enter(p)$ models the process entering the critical section while event $leave(p)$ models the process leaving the critical section. We add a task-based deadline constraint for

```
invariants
  @inv1 wait ⊆PROCESS
  @inv2 process ⊆PROCESS
  @inv3 finite(process)
  @inv4 card(process)≤ 1
  @inv5 wait ∩process = ∅
event wish
  any p
  where
    @grd1 pro∈ PROCESS\wait
    @grd2 pro∈ PROCESS\process
  then
    @act1 wait:= wait∪{p}
end
```

```
event enter
  any p
  where
    @grd1 pro∈ wait
    @grd2 process = ∅
  then
    @act1 wait:= wait\{p}
    @act2 process:= process ∪{p}
end

event leave
  any p
  where
    @grd1 pro∈ process
  then
    @act1 process:= process\{p}
end
```

Fig. 2: Abstract Model With Timed Mutual Exclusion Problem

each process in the first refinement, which ensures that if a process wishes to enter its critical section, it will enter the critical section within a certain deadline. The specification of the task-based deadline is presented in (2), which states that any process that wishes to enter the critical section, will enter the critical section within ddl. Figure 3 shows the refinement with task-based deadline for each process. $t1(p)$ models the timestamp at which process p wishes to enter the critical section. $r1(p)$ models the timestamp process entering the critical section. $@inv4$ captures the task-based deadline constraint, and $@inv5$ is needed to prove that $@inv4$ is preserved. $@grd1$ of $Tick$ event ensures $@inv5$ is preserved.

$$\forall p \cdot p \in wait \;\Rightarrow\; Deadline(wish(p), enter(p), ddl) \tag{2}$$

```
invariants
  @inv1 clk ∈ ℕ
  @inv2 t1 ∈ wait →0..clk
  @inv3 r1 ∈ process →0..clk
  @inv4 ∀p·r1(p)>t1(p) ⇒r1(p)−t1(p
        )≤ ddl1
  @inv5 ∀p·p∈ wait ⇒clk−t1(p)≤
        ddl1

event enter extends enter
  then
    @act3 r1(p):= clk
end
```

```
event wish refines wish
  any p
  where
    @grd1 pro∈ PROCESS\wait
    @grd2 pro∈ PROCESS\process
  then
    @act1 wait:= wait∪{p}
    @act2 t1(p):= clk
end

event tick
  where
    @grd1 ∀p·p∈ wait ⇒clk+1−t1(p)
          ≤ ddl1
  then
    @act1 clk:= clk+1
end
```

Fig. 3: First refinement with task-based deadline constraint

4 Scheduler-Based Deadline Constraint with Nondeterministic Queue Based Scheduling

In concurrent computing, concurrent processes are executed by interleaving the execution steps of each process, which models processes in the outside world that happen concurrently. In real time systems, scheduling is used to make sure that all processes meet their deadlines [4]. A scheduler is used to allocate the resource to a process for some time.

In this case study, we specify two scheduler-based deadlines: (3a) and (3b). (3a) requires that when the system is idle, one of the requesting processes will enter the critical section within *ddl*3. Specifically, there are two cases that trigger the scheduling of the enter event: 1) a process wishes to enter and both the queue and the critical section are empty, and 2) some process leaves the critical section and there is some other process waiting in the queue. Observe here that events can act as timing triggers only under certain conditions, e.g., the wish event is only a timing trigger when the queue and critical section are empty. To deal which such conditional triggers, we split the event into separate refinements representing separate cases. We refine the *wish* event into a *wish_empty* event, enabled when condition 1 is true, and a *wish_nonempty* event, enabled in all other cases. Similarly, we split the *leave* event into a *leave_nonempty* event, enabled when condition 2 is true, and a *leave_idle* event, enabled in the other cases (see Fig. 5). The events *wish_empty* and *leave_nonempty* are therefore used as trigger events in (3a), whereas the response event *enter* is the event modeling entering the critical section.

(3b) requires that, once a process enters the critical section, it will leave the critical section within *ddl*2. Therefore the trigger event is the *enter* event, whereas the response events should correspond to leaving the critical section. As the latter is now captured by *two* events, there are two response events in (3b).

$$Deadline(\{wish\_empty, leave\_nonempty\}, enter, ddl3) \tag{3a}$$

$$Deadline(enter, \{leave\_empty, leave\_nonempty\}, ddl2) \tag{3b}$$

To refine the task-based deadline constraint with scheduler-based deadline constraints, we propose a nondeterministic queue based scheduling framework to address the schedule order of the sequential execution of a set of events. In this framework, a queue is used to manage the ready processes. Each process is assigned a position in the queue, formally: $queue \in wait \rightarrowtail (0..N-1)$. When one process is ready, it is nondeterministically assigned a natural number that is not in the range of the queue. Only the process in the front of the queue ($min(ran(queue))$) can get the resource to run. The dequeue operation will decrease the indexes of all the other processes in the queue by the index of the front process plus one ($min(ran(queue))+1$) to guarantee that once a process is added to the queue, it will eventually get the chance to run. In the second refinement, we use this nondeterministic queue based framework to impose an order on the execution of the concurrent tasks. This refinement prevents a process from entering the critical section forever without allowing other processes to enter the critical section. The second refinement is shown in Figure 4.

```
invariants
  @inv2 queue ∈ wait ↣ 0..N−1
event wish extends wish
  any i
  where
    @grd3 i ∈ 0...N−1
    @grd4 i∉ ran(queue)
  then
    @act3 queue(p):= i
end
```

```
event enter extends enter
  any j
  where
    @grd4 p=queue∼(j)
    @grd5 j=min(ran(queue))
    @grd6 j∈ ran(queue)
    @grd7 queue≠∅
  then
    @act4 queue:= (λ q·q∈ dom({p}⩤
          queue)| queue(q)−j−1)
end
```

Fig. 4: Second Refinement with Queue Based Scheduling Framework

In order the prove that the scheduler deadlines refine the task-based deadlines, invariants capturing the relation between task-based deadlines and scheduler-based deadlines are needed. Assume that in the abstract machine the trigger event of one process p occurs at timestamp $t1$, and the deadline is ddl. In the refined machine the trigger event ($wish\_empty/wish\_nonempty$) starts at timestamp $t3$ and its deadline is $ddl3$. The trigger event ($enter$) starts at timestamp $t2$ and its deadline is $ddl2$. We assume that all processes have the same maximum possible deadline to enter and leave the critical section $ddl23 = ddl2 + ddl3$. The process p has to wait for all the processes ahead of it in the queue to enter and leave the critical section. The total waiting time is proportional to its index in the queue, which is $queue(p) * ddl23$. If the critical section is empty and the time that last process leaves the critical section is $t3$, p should enter the critical section within $t3 + queue(p) * ddl23 + ddl3$. If the critical section is not empty and the time that last process enters the critical section is $t2$, p should enter the critical section within $t2 + queue(p) * ddl23 + ddl23$. Based on different conditions, the sum of the refined sequential deadline should be less than abstract deadline $t1 + ddl1$, which is shown in @$inv9$ and $inv10$ in Figure 6. @$inv9$ and $inv10$ present these two conditions as required gluing invariants. Assume there are N processes, the worst case is $N - 1$ processes in the waiting list. As @$axm8$ presents, in the worst case the refined deadline is less than the abstract deadline. Figure 6 shows the required axioms and invariants to refine the task-based deadlines to scheduler-based deadlines. @$inv5$ and @$inv8$ present the invariant for scheduler-based deadlines (3b), @$inv6$ and @$inv7$ present the invariants for scheduler-based deadlines (3a).

5 Two Implementation of Nondeterministic Queue-Based Framework

The nondeterministic queue based scheduling framework is a general framework that assign indexes to processes nondeterministically. By applying additional rules on the assignment of these indexes, the queue based scheduling framework can be refined to some scheduling policies such as First In First Out (FIFO) and deferrable priority based scheduling policy with aging technique.

First In First Out FIFO is one of the scheduling policies that guarantee that the resources are assigned to each process with the order that they require the resource. The FIFO scheduling policy handles all processes without priorities. The queue based scheduling framework assigns each process with a corresponding natural number $k \in \mathbb{N}$, and FIFO scheduling policy limits this natural number to the current size of the queue. And when the critical section is empty, the process that is in the front of queue leaves the queue and enters the critical section. The indexes of all the other processes in the queue are reduced by one.

The refinement from the scheduler-based model is shown in Figure 7. Initially the queue is empty and $qsize$ is zero. Whenever some process is added to the queue, it is assigned with the number of queue size and the queue size increases

```
event wish_empty extends wish
where
  @grd5 wait=∅∧process=∅
then
  @act5 t3 := clk
end

event wish_nonempty extends wish
where
  @grd5 wait≠∅∨process≠∅
end

event enter extends enter
then
  @act6 t2 := clk
  @act7 r3 := clk
end
```

```
event leave_nonempty extends leave
where
  @grd2 queue≠∅
then
  @act3 r2 := clk
  @act4 t3 := clk
end

event leave_idle extends leave
where
  @grd2 queue=∅
then
  @act2 r2 := clk
end

event tick refines tick
where
  @grd2 process=∅ ∧wait≠∅ ⇒clk
        +1−t3≤ ddl3
  @grd4 process≠∅ ⇒clk+1−t2≤ ddl2
then
  @act1 clk:= clk+1
end
```

Fig. 5: Third Refinement to scheduler-based deadline Constraint

```
axioms
  @axm7 ddl23 = ddl2+ddl3
  @axm8 ((N−1) ∗ ddl23)+ddl3 ≤ ddl1
invariants
  @inv5 r2>t2 ⇒r2−t2≤ ddl2 // deadline2( enter, leave_idle|leave_nonempty,
        ddl2)
  @inv6 r3>t3 ⇒r3−t3≤ ddl3 // deadline3( wish_empty|leave_nonempty, enter,
        ddl3 )
  @inv7 queue≠∅ ∧process=∅ ⇒clk−t3≤ ddl3 // required for enter to preserve
        inv6
  @inv8 process≠∅ ⇒clk−t2≤ ddl2 // required for leave to preserve inv5
  @inv9 ∀p·process=∅ ∧p∈ wait ⇒t3+(queue(p)∗ddl23)+ddl3 ≤ t1(p)+ddl1
  @inv10 ∀p·process≠∅ ∧p∈ wait ⇒t2+(queue(p)∗ddl23)+ddl23 ≤ t1(p)+ddl1
```

Fig. 6: Axioms and invariants needed to refine task-based deadline to scheduler-based deadline

```
event enter refines enter
  any p j
    where
    @grd1 pro∈ wait
    @grd2 process = ∅
    @grd6 j∈ ran(queue)
    @grd4 p=queue∼(j)
    @grd5 j=0
    @grd7 queue≠∅
  then
    @act1 wait:= wait\{p}
    @act2 process:= process ∪{p}
    @act3 r1(p):= clk
    @act4 queue:= (λ q·q∈ dom({p}⩤
           queue) | queue(q)−j−1)
    @act6 t2 := clk
    @act7 r3 := clk
    @act8 qsize:= qsize−1
end
```

```
invariants
  @inv1 qsize ∈ 0..N−1
  @inv2 ∀i·i≥ qsize⇒i∉ ran(queue)

event wish_empty refines wish_empty
  any p i
  where
    @grd1 pro∈ PROCESS\wait
    @grd2 pro∈ PROCESS\process
    @grd3 i =qsize
    @grd5 wait=∅∧process=∅
    @grd6 qsize<N
  then
    @act1 wait:= wait∪{p}
    @act2 t1(p):= clk
    @act3 queue(p):= i
    @act5 t3 := clk
    @act6 qsize:= qsize+1
end
```

Fig. 7: Refinement with FIFO Queue

by one. When the critical section is empty, the process in the front of queue $queue(0)$ is removed from the queue, the indexes of all the other processes in the queue are reduced by one. The queue size also decreases by one. *wish_nonempty* uses the same refinement strategy as *wish_empty*.

Deferrable Priority Based Scheduling with Aging Technique Fixed priority scheduling policies assign the tasks with fixed priorities: $priorities \in event \to \mathbb{N}$. The scheduler will select the tasks with higher priorities to access the system resources before those with lower priorities. However there is a disadvantage of these scheduling policies that tasks with lower priorities may be starved when the tasks with higher priorities keep coming and jumping the queue. An aging technique is used to ensure that tasks with lower priorities eventually complete their execution. The general way to implement aging technique is to increase the priorities of the tasks with lower priorities while they are waiting in the ready queue. However with the increasing priorities of some processes, it will occupy the positions of some other processes. Deferrable priority based scheduling allows that when the position of one process is occupied by some other processes, this process is deferred with a random position after its assigned position. Using the timed mutual exclusion problem as an example, our approach to refine the scheduling framework to priority based scheduling with aging technique is to define a $pindex \in PROCESS \rightarrowtail\!\!\!\!\twoheadrightarrow 0..N-1$, where $pindex$ is a bijection function from $PROCESS$ to natural numbers. Equation (4) shows that the higher priority of a process is, the lower its index is.

$$\forall a, b \cdot a \in PROCESS \wedge b \in PROCESS \wedge priority(a) < priority(b) \\ \Rightarrow pindex(a) > pindex(b) \tag{4}$$

To avoid the starving problem of processes with lower priorities, we add a rule to the priority based scheduling that when the position of some process is occupied by some other process with lower priority, which means that the lower priority one has waited some time in the queue, the high priority one is deferred by some higher random index. Specifically, the indexes of the processes are decreasing by $min(ran(queue)) + 1$ when the process at the front queue, whose index is $min(ran(queue))$, leave the queue and enter the critical section. The *enqueue* operation will assign the process with its corresponding index in the queue. However, this would cause a conflict as this operation will make some processes occupy the spaces of some other processes. For example, process a's level is 3 and process b's level is 2. One process c is at the front of queue. When c leaves the queue, the index of a is reduced to 2. When b wishes to enter the queue, its position is taken by a. Here we choose the next available space available in the queue $i = min(k \mid k \in ran(pindex) \wedge k \notin ran(queue) \wedge k > pindex(p))$. When the position is not taken by other processes, the process takes its assigned position $pindex(p)$. The dequeue operation is the same as the basic queue based scheduling framework. Figure 8 shows the refinement from scheduler-based model with a deferrable priority based scheduling policy with aging technique.

```
event wish_empty refines wish_empty
  any p i
  where
    @grd1 pro∈ PROCESS\wait
    @grd2 pro∈ PROCESS\process
    @grd3 {k|k∈ ran(pindex)∧k∉ ran(queue)∧k≥ pindex(p)}≠∅
    @grd4 i=min({k|k∈ ran(pindex)∧k∉ ran(queue)∧k≥ pindex(p)})
    @grd5 wait=∅∧process=∅
  then
    @act1 wait:= wait∪{p}
    @act2 t1(p):= clk
    @act3 queue(p):= i
    @act5 t3 := clk
end
```

Fig. 8: Refinement with Deferrable Priority Based Scheduling with Aging Technique

Proof Statics Table 1 shows the proof statics of the model. Here $m0$ is the abstract machine with a simple mutual exclusion problem. $m1$ refines $m0$ with the task-based deadline for each process. $m2$ refines $m1$ with a nondeterministic queue based scheduling policy. $m3$ refines $m2$ to the scheduler-based deadline with additional gluing invariants. $m4\_fifo$ and $m4\_priority$ are two different implementations of the nondeterministic queue. FIFO queue has more proof obligations than priority queue because additional invarinats @$inv1$ and @$inv2$ of Figure 7 are needed for the FIFO queue refinement.

Table 1: Proof Statistics

Machine	Number of Generate PO	Automatically Proved	Automatically Proved%
m0	7	7	100
m1	18	18	100
m2	9	7	77.8
m3	49	43	87.8
m4_fifo	17	12	70.6
m4_priority	9	7	77.8

6 Conclusions and Future Work

Based on a trigger-response approach to modeling deadlines in Event-B, we distinguish the concept of task-based timing properties and scheduler-based timing properties from the perspective of different system design phases. We define timing properties that place discrete time constraints on individual processes or tasks as task-based timing properties, which describe high level timing properties from system requirement specification phase. These task-based timing properties can not describe the concurrent behaviour of tasks precisely. In real time systems, schedulers are used to schedule concurrent tasks. To model the behaviour of these concurrent processes, we define scheduler-based timing properties as concrete timing properties for the system design phase, which place discrete time constrains on the scheduler which schedules the concurrent tasks. To refine task-based timing properties to scheduler-based timing properties, we introduce a nondeterministic queue based scheduling policy with some additional gluing invariants. The queue based scheduling policy can also be implemented as a FIFO queue scheduling policy or a deferrable priority based scheduling policy with aging technique. We prove that the two refinements of the nondeterministic queue satisfy the same deadlines with the mutual timed exclusion case study.

This paper does not address the possible time deadlock caused by the trigger-response pattern. For example, if the delay is larger than the deadline between the same trigger-response pair, there would be a point where the global clock cannot proceed as it is constrained by the deadline constraint not to proceed but also constrained by the delay constraint to proceed, a deadlock will occur in the model. Additional conditions to avoid these deadlocks and formal specifications of the enabledness and weak fairness assumptions on response events with proofs are left for future work. In addition, the mutual exclusion case study assumes no intermediate events between trigger and response events, while intermediate events are common in real systems such as CPS. More exploration is needed for the enabledness of intermediate and response events under different situations. Fairness and convergence assumptions on intermediate events and response events will help with the scaling of the proposed approach.

In the cases that the system does not require explicit mention of time, the notion of bounded fairness and finitary fairness allows one to express eventual occurrence of a set of events. Some work has been done to model fairness in Event-B [17,18]. Bounded fairness modeling as well as finitary fairness modeling can be researched further with some addition prove rules and refinement frameworks.

In order to explicitly represent timing properties in a cyber physical system, there are three typical time constraints to look into: period, deadline, worst-case execution time. More work can be done to apply some scheduling policies such as Rate-Monotonic (RM) and priority inheritance protocol based on the queue based scheduling framework to analyze real-time performance of CPS together with the mentioned time constraints in Event-B.

Acknowledgement This work is supported in part by the scholarship from China Scholarship Council (CSC) under the Grant CSC NO. 201708060147.

References

1. Abrial, J.R.: The B-book: assigning programs to meanings. Cambridge University Press (2005)
2. Abrial, J.R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press (2010)
3. Alur, R., Henzinger, T.A.: Finitary fairness. In: Proceedings Ninth Annual IEEE Symposium on Logic in Computer Science. pp. 52–61 (Jul 1994)
4. Alur, R.: Principles of Cyber-Physical Systems. The MIT Press (2015)
5. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical computer science 126(2), 183–235 (1994)
6. Banach, R., Butler, M., Qin, S., Verma, N., Zhu, H.: Core hybrid event-b i: Single hybrid event-b machines. Science of Computer Programming 105, 92 – 123 (July 2015)
7. Butler, M.: Mastering System Analysis and Design through Abstraction and Refinement. IOS Press (2013), http://eprints.soton.ac.uk/349769/
8. Butler, M., Abrial, J.R., Banach, R.: Modelling and refining hybrid systems in Event-B and rodin. In: Petre, L., Sekerinski, E. (eds.) From Action System to Distributed Systems: The Refinement Approach. Taylor & Francis (April 2016), https://eprints.soton.ac.uk/376053/
9. Butler, M., Falampin, J.: An approach to modelling and refining timing properties in B (January 2002), https://eprints.soton.ac.uk/256235/
10. Cansell, D., Méry, D., Rehm, J.: Time constraint patterns for event b development. In: Julliand, J., Kouchnarenko, O. (eds.) B 2007: Formal Specification and Development in B. pp. 140–154. Springer Berlin Heidelberg (2006)

11. Dershowitz, N., Jayasimha, D.N., Park, S.: Bounded Fairness, pp. 304–317. Springer Berlin Heidelberg (2003), https://doi.org/10.1007/978-3-540-39910-0_14
12. Dierks, H., Kupferschmid, S., Larsen, K.G.: Automatic abstraction refinement for timed automata. In: Raskin, J.F., Thiagarajan, P.S. (eds.) Formal Modeling and Analysis of Timed Systems. pp. 114–129. Springer Berlin Heidelberg (2007)
13. Graf, S., Prinz, A.: Time in state machines. Fundamenta Informaticae 77, 143–174 (2007)
14. Jastram, M., Butler, P.: Rodin User's Handbook: Covers Rodin V.2.8. 2.8covers Rodin, Createspace Independent Pub (2014), https://books.google.co.uk/books?id=ws2WoAEACAAJ
15. Larsen, K.G., Pettersson, P., Yi, W.: Uppaal in a nutshell. International journal on software tools for technology transfer 1(1-2), 134–152 (1997)
16. Sarshogh, M.R., Butler, M.: Specification and refinement of discrete timing properties in Event-B. AVoCS 2011 (2011), https://eprints.soton.ac.uk/272480/
17. Sekerinski, E., Zhang, T.: Finitary fairness in event-b. In: Dagstuhl Seminar on Refinement Based Methods for the Construction of Dependable Systems (Dagstuhl, Germany (2009)
18. Sekerinski, E., Zhang, T.: Finitary fairness in action systems. In: Liu, Z., Woodcock, J., Zhu, H. (eds.) Theoretical Aspects of Computing – ICTAC 2013. pp. 319–336. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
19. Sulskus, G., Poppleton, M., Rezazadeh, A.: An interval-based approach to modelling time in Event-B. Fundamentals of Software Engineering 9392, 292–307 (2015), http://eprints.soton.ac.uk/377201/