Timebands Framework – 2011

Alan Burns, Kun Wei, Jim Woodcock
Department of Computer Science,
University of York, UK.

Ian J. Hayes
University of Queensland,
Australia

Cliff B. Jones
Department of Computer Science,
University of Newcastle, UK.

Abstract

Complex real-time systems must integrate physical processes with digital control, human operation and organisational structures. New scientific foundations are required for specifying, designing and implementing these systems. One key challenge is to cope with the wide range of time scales and dynamics inherent in such systems. To exploit the unique properties of time, with the aim of producing more dependable computer-based systems, a timeband framework has been developed. In this paper we review this framework and report on modifications, extensions and applications that have been undertaken in funded research that completed in 2011.

1 Introduction

One characteristic of complex computer-based systems is that they are required to function at many different time scales (from microseconds or less to months or more). Time is clearly a crucial notion in the specification (or behavioural description) of such systems, but it is usually represented in modeling schemes for example as a single flat physical phenomenon. Such an abstraction fails to support the structural properties of the system, forces different temporal notions on to the same flat description, and fails to support the separation of concerns that the different time scales of the system facilitate. Even with a single time scale, system architects seem to have great difficulty in specifying temporal properties in anything other than very concrete implementation-level terms. But just as the functional properties of a system can be modelled at different levels of abstraction or detail, so too should its temporal properties be representable in different, but provably consistent, time scales.

To make better use of 'time', with the aim of producing more dependable computer-based systems, a framework has been developed that explicitly identifies a number of distinct *time bands* in which the system under study is situated. The *Timebands* framework enables observations of existing systems to be described and the requirements for new or modified systems to be specified.

An initial description of the framework was developed in 2005 [7, 5, 3]. Further developments, and formalisation, of the framework were then undertaken under the auspices of the EPSRC funded project, INDEED, which ran from January 2007 to February 2011. This report reviews the findings of this project and provides an update on the definition of the Timebands framework that reflects the project's activities and findings. In addition to these activities a detailed case study involving a system of systems was undertaken as part of industry-based PhD work undertaken by White [41]. Also addressing, amongst other topics, issues relating to the Timebands framework is the project *Time Bands for Teleo-Reactive Programs* undertaken at the University of Queensland and funded by the Australian ARC.

This report has the following structure. Sections 2 to 5 report on aspects of the framework that have been modified. Section 6 then gives an overview of the work undertaken to formalise the framework. Section 7 then looks at some of the issues involved when both discrete and continuous behaviours are present. This is followed by a discussion on tools and case studies. Finally in Section 9 we presents some conclusions and summaries future work.

2 Overview of the Timebands Framework

In the Timebands framework a system is assumed to consist not of a single time dimension but a finite set of partially ordered *bands*. Each band is represented by a *granularity* (expressed as a unit of time that has meaning within the band, e.g. the minute band) and a *precision* that is a measure of the accuracy of the time frame defined by the band. System activities are placed in some band **B** if they engage in significant events at the time scale represented by **B**. They have dynamics that give rise to changes that are observable or meaningful in band **B**'s granularity.

A band is populated by *events* and *activities*. Events, in the normal way, are considered to be instantaneous (a cut of the time line) within the band of their definition. Activities by comparison have duration of one or more 'units' of the band's granularity. Within an activity there will be collections of events with precedence constraints defined to represent the behaviour of the activity. The start of an activity will be an event (that precedes all others) and the end of an activity will similarly be an event (that is preceded by all other events in the activity).

Behaviours across bands are formulated as *mappings* between events in one band and activities in a 'lower' band. So, events that are instantaneous in one band may *map* to activities that have duration at some lower band with a finer granularity.

An illustration of a three band system with the mapping of events to activities is shown in Figure 1. As noted, the start and end of an activity are themselves represented as events that may themselves map to lower bands. In this illustration, the duration of activity X in band **B** must be less than the precision of band **A**; similarly the duration

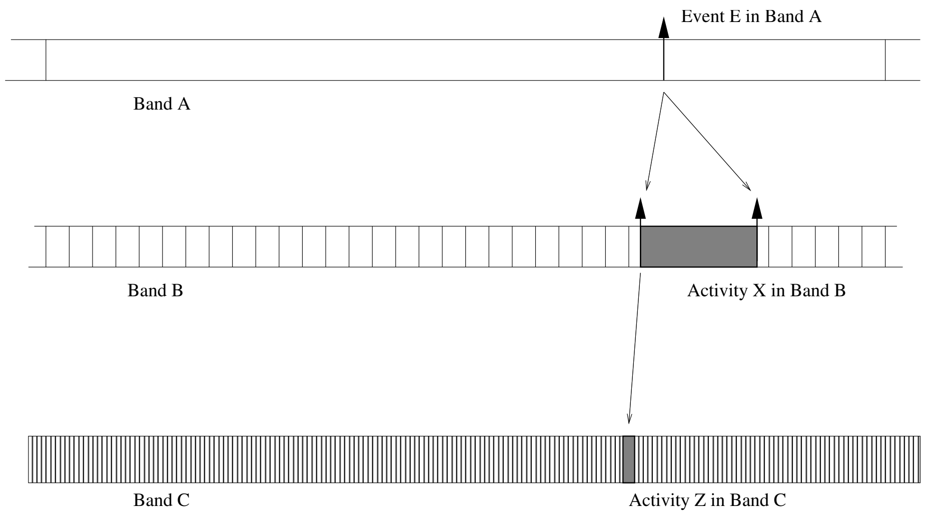

Figure 1: Time Band Example

of activity Z in band **C** must be less than the duration of band **B**.

One of the motivations for the framework is the recognition that the dynamics (rates of change) of system components form an effective means of structuring complex systems. A hierarchy based on bands of time allows natural patterns of behaviour to become apparent. Activities require agents and resources to be scheduled and managed, and this occurs at all bands within the system. Many systems will contain repetitive behaviour at all levels from yearly reviews, weekly schedules, hourly plans, minute by minute sampling and below that levels of control at the 10s of millisecond level. Hardware platforms then operate at even finer time scales. Cascade control, for example, is naturally described as embedded control loops each an order of magnitude slower than the next. The Timebands framework allow these forms of *self-symmetry* to be expressed in a structured way.

It is important to emphasise that the full behaviour of a system is not obtained by refining down to the lowest band or by projecting emergent behaviours up to the highest band. Rather it is the amalgamation of all band descriptions – all have behaviours that may be needed in any assertion about the system as a whole.

The current, most complete, published description of the Timebands framework is contained in the paper by Burns and Hayes [6]. The key notions that are central to the framework are summaries as follows:

- *time band* – a subset of system behaviours (discrete and continuous) with similar temporal properties;
- *system* – a partially ordered set of bands;
- *separation* – the property of being able to assume that activities in lower (quicker) bands are instantaneous and the state of higher (slower) bands is unchanging;

- *granularity* – the unit of time defined by a band;
- *precision* – the constraint on instantaneous behaviour within a band (a bound on the allowed imprecision as to when an event occurs);
- *event* – an instantaneous behaviour within a band;
- *activity* – an behaviour with duration within a band;
- *duration* – a time interval between two events in the same band, expressed as an integer number of granularities of the band;
- *clock* – an abstract band-specific series of ticks (events) at the granularity of the band;
- *precedence* – one event preceding another event;
- *simultaneous* – two events occurring at the same instant;
- *immediate* – a precedence relation between two simultaneous events;
- *mapping* – a link between a set of events in one band and a set of activity in a lower band;
- *state* – observable variables available within a specific time band;
- *change-of-state events* – for discrete observations;
- *accuracy* – maximum "instantaneous" change in a continuous variable in the context of a defined band;
- *rate-of-change* – maximum rate of change of a continuous variable over a band's granularity.

The last two notions are band-specific, they denote the maximum change there can be to a continuously changing external entity during the granularity and precision of the band. An external entity can be observed in more than one band; in which case there will be accuracy and rate-of-change parameters defined for each of these bands. Note such values will usually be expressed as assumptions as to the behaviour of the environment.

2.1 Intuitive Examples

A number of illustrative and intuitive examples were used to exercise the framework and to evaluate the expressive power of different means of formalizing the model. One larger example, the mine pump system, is described in Section 8. Another example is that of a university lecture course. Here there are immediately four bands to identify: the Year band in which new courses and curriculum are planned, the Week band in which lectures are scheduled, the Minute band that allows each lecture to be structured, and the Second band that can capture various interactions with the available technical support (e.g., laptop response).

Other smaller examples include giving precise meanings (and scope for implementation) to:

- Fridge light – “light must come on immediately the door is open”.
- Security door – “must remain closed at all times”.
- Top and Bottom sensors – “must never be on at the same time”.
- Clock chimes – how to relate the 12 chines at midday with the specific time of 12 o’clock.
- temporal agreement – “meeting at 12 o’clock”.

3 Simultaneous and Overlapping Events

Many of the areas that required clarification during the formalisation of the framework related to the relation between groups of events in one band and their associated (mapped) activities in another finer band. One of these issues relates to precedence: if event e is before event f, what constraints are there on the activities E (which is mapped to e) and F (which is mapped to f)? This issue is not as straightforward as it may initially seem – and is discussed further in Section 4.

Another aspect of the framework’s definition that needed to be completed during the formalisation activity was the definition of *simultaneous*. When event e in band **B** is mapped to activity E in band **C** the timeband framework imposes a constraint that the duration of activity E in band **C** must be less then the precision of band **B**. This supports the notion that an instantaneous phenomenon in one band can be given behavioural structure in another band as long as the duration on this behaviour is bounded by a small value (the *bound on instantaneousness*). So for example in a band of granularity of one second, the precision may be 10ms; an event can be regarded as instantaneous in this band if the duration of the activity it maps to is less than 10ms.

This notion of an instantaneous event is extended in the timebands framework to include two or more simultaneous events. Events can be simultaneous and they can also be ordered, in which case we say that one event immediately follows the other. The *simultaneous* constraint on two simultaneous events is that their activities must both be fully contained within an interval of maximum duration defined by the precision of their band. So from the earliest start of either activity to the latest end, again, of either activity is bounded by precision.

Note the notion of simultaneous is not transitive. Events e and f can be simultaneous, as can events f and g; but this does not imply that e and g are simultaneous.

During the explorative case studies it became clear that the property of ‘not simultaneous’ is open to two interpretations, both of which are useful. Two activities can be close to each other, but not sufficiently close for their mapped events to be considered simultaneous. Alternatively they can be further apart, with a gap of at least the precision of the band. We define the term *separated* to mean that there is an interval of duration of at least the precision between the end of the ‘earlier’ activity and the start of the ‘later’ one. More precisely, two events, e and f, are separate if there could not exist an event *h* that is both simultaneous with *e* and simultaneous with *f*. An example of separate events are clock ticks, they are regular but never close to each other.

It follows from the definition of *separate* that two events could be neither separate nor simultaneous. In this situation the mapped activities are close to each other, perhaps overlapping, but are not simultaneous (i.e., not fully contained within the precision).

A third state could be defined (as not separate and not simultaneous) but the intuition for such a notion is unclear. Rather it seems more useful to define a third term - two events *coincide* if they are not separate. It follows that that two simultaneous events also coincide, but two events that coincide are not necessarily simultaneous.

4 Precedence and Signature Events

The issue of temporal order, addressed in the Framework, gives rise to a number of possible interpretations as to the notion of precedence; put simply "what constraints should there be on the internal events of activities E and F if e maps to E and f maps to F, and e is defined to precede f?".

First we introduce some required notation. Only two bands are needed to describe the consistency issues, these will be called the *High* and *Low* bands. The *High* band has the coarser granularity. Events is this band are represented by lower case Greek letters (e.g., α and β). Events in the *Low* band use ordinary lower case (e.g., e and f) and activities in this band are referenced by upper case (e.g., A and B). There are however three special events within each activity (eg, A): the start event ($a\!\uparrow$), the end event ($a\!\downarrow$) and the signature event ($\hat{a}$) – the later being used in Section 6 to help in the formalization of the framework. Note these symbols are just names of events (not operators).

In a complete model events and activities will repeat; a specific event (object) is thus defined by an event type (or class) and an occurrence of the event. Occurrences will be ordered, and many repeating events (such as the notional ticks of a clock) will be separate (see definition above). *In the following discussions however, the term event will mean a specific occurrence of an event type.*

4.1 Ordering of Events

As events are instantaneous then the definition of precedence is relatively straightforward. Starting with strict order ($\prec$), this is transitive and reflexive: $e \prec f$ and $f \prec g$ implies $e \prec g$), and $e \prec f$ and $f \prec e$ is false (i.e., not possible; neither is $e \prec e$). All events in an ordering relation must be from the same band. As all activities have duration it follows that $a\!\uparrow \prec a\!\downarrow$.

If we move away from strict order to define precedence ($\preceq$) then if we have $e \preceq f$ and $f \preceq e$ there are two possible interpretations:

- e and f are equal (same event)
- e and f are equivalent (not same event)

These define a *partial order* or a *preorder*, respectively. In this treatment we shall use preorder; so two arbitrary events from the same band are:

- unrelated,
- ordered, or
- equivalent.

In all definitions of precedence there is transitivity. This is needed to give order to specifications. For example, all events that occur before 12 o'clock are separated from (and prior to) all events that happen after 13 o'clock. This is because clocks behave monotonically and the basic predicate 'before/after' is defined to be transitive. We shall note below that with granulated time, transitivity is not supported automatically.

4.2 Interleaving intervals for Activities

Activities have duration (their behaviour exists for an interval of time) and there is therefore more than one way that two activities (two time intervals) can be aligned. Consider activities A and B, the following are all possible:

- All of A is before all of B: $a\downarrow \preceq b\uparrow$
- Some of A is before all of B: $a\uparrow \preceq b\uparrow$
- All of A is within B: $b\uparrow \preceq a\uparrow \prec a\downarrow \preceq b\downarrow$
- Some of A is before some of B: $a\uparrow \preceq b\downarrow$

This can be continued to give symmetric definitions for when B is before A in some form or another.

To complete the notation we define the mapping relation ($\longrightarrow$) to act between an event (α) in band X and an activity (A) in band Y (with the necessary finer granularity): $\alpha \xrightarrow{X,Y} A$.

4.3 Precedence and Mapping

To return to the central question. Given events α and β with $\alpha \preceq \beta$ and $\alpha \xrightarrow{high,low} A$, $\beta \xrightarrow{high,low} B$, what is the required relation between A and B? In the following a number of different schemes are outlined, each with the positive aspects of the scheme (pros) and the drawbacks (cons). Note there are some minimal requirements where the intuition is clear:

$a\downarrow \preceq b\uparrow \Rightarrow (\alpha \preceq \beta \wedge a\uparrow \preceq b\downarrow)$

and

$a\downarrow \prec a\uparrow \Rightarrow (\alpha \prec \beta \wedge a\uparrow \prec a\downarrow)$

Strongest constraint: All of A is before all of B ($a\downarrow \preceq b\uparrow$).
Pros: Clearly furnishes transitivity.
Cons: Too constraining, requiring all of A to complete before B can proceed. In many examples A will have internal events that need not occur before B can start – for example the events *door-open* and *light-on* may be linked by precedence, but the mapped

activities do not want to be constrained so that the door has to be fully open (activity complete) before the turning light on activity can start.

Weakest Constraint: There exist some event in A that is before some event in B ($\exists\, a, b : a \in A, b \in B, a \preceq b$)
Pros: Maximizes non-determinacy. Defines precedence recursively through pairs of events within each band.
Cons: Overlapping activities (A and B) would be consistent with $\alpha \preceq \beta$ and $\beta \preceq \alpha$, forcing an equivalence between α and β. Moreover the constraint does not ensure transitivity.

Constraining Starts: Require B to start after A has started ($a\!\uparrow\, \preceq b\!\uparrow$)
Pros: Provides transitivity.
Cons: Too constraining, A and B may be started concurrently and it is not necessary for these starts to be constrained. Activity A may start later but be guaranteed to progress quicker and be able to reach whatever position is needed to meet the precedence constraint.

Constrained Ends: Require B to finish after A has finished ($a\!\downarrow\, \preceq b\!\downarrow$)
Pros: Provides transitivity.
Cons: Again too constraining and fails to map on to application requirements, ends of activities are independent of each other even if there is a precedence relation between some aspect of their behaviour.

Constrained by Signature Events: ($\hat{a} \preceq \hat{b}$)
Pros: In many situations it is natural to represent an entire activity by a single signature event. There is then a natural affinity between the mapped event (α) in band *high* and the signature event ($\hat{a}$) within activity A in band *low*. Definition also provides transitivity.
Cons: It is too restrictive to allow an activity to only have one signature event. The activity may be involved in two or more key interactions, each of which could reasonable be declared the signature event.

Constrained by Multiple Signature Events: ($\exists\, \hat{a_x} \in A, \hat{b_y} \in B, \hat{a_x} \preceq \hat{b_y}$)
Pros: Flexibility: allows for more than one signature event per activity.
Cons: Does not provide transitive.

Constrained by Communication Events: ($\exists\, write \in A, read \in B, write \prec read$)
Pros: Allows precedence to be defined only when there is a concrete communication between the activities.
Cons: As there can be more than one communication event between two activities, suffers the same problems as 'weakest constraint', in particular lose of transitivity.

Constrained Two Phase Behaviour:
($\exists\, write \in A, read \in B, write \prec read \;\wedge \forall\; write, read \in A/B, read \preceq write$)
Pros: Here all communication is constrained so that reads occur before writes within the same activity. This provides for transitivity. Two phase behaviour could also be defined via the ordering of signature events.

Cons: Restricts all activities to have a particular structure.

Constrained by Time: In some formalism, precedence is defined in terms of time, $e \preceq f$ implies, for an abstract clock function *time* that is defined for events, $\mathit{time}(e) \leq \mathit{time}(f)$. Note this formulation is not used in the Timeband framework where the abstract clock in any band only counts at the granularity of the band (e.g., seconds for the 'seconds' band). In the lower band the 'time' of an activity ($T(A)$) could be defined as $(\mathit{time}(a\uparrow) + \mathit{time}(a\downarrow))/2$, i.e. the mid point time of the activity. The precedence constraint could then be defined as $T(A) \leq T(B)$.
Pros: Provides for transitivity
Cons: The middle point is arbitrary (as would be any other point) and does not relate to the functionality of the activity from which the precedence relations is derived.

To summarise, the requirement for transitivity forces some quite strong restrictions on the structure of activities such as strict ordering, a single signature event or two-phase behaviour. Looser restrictions such as the weak constraint model and the multiple signature events model are naturally more expressive but fail to provide for that most natural property of transitivity over precedence/order.

As none of the schemes outlined above seem to give a universal structure to link precedence and mappings between bands, it is perhaps necessary to provide a more context-specific definition of consistence (between the bands).

A complete specification involves the union of the behaviours in all bands. However it is possible to consider a top-down or bottom-up development of a system specification. Top-down has some of the properties of refinement (event refinement), and bottom-up is concerned with emergent (or derived) behaviour.

For top-down, the precedence relationship between the events is specified first. When the activities are specified then it would be required to explicitly state the consistency condition. This could use any of the schemes outlined above that support transitivity, for example the use of a single signature event. But there is no single scheme 'that fits all situations'. In the Timebands framework two events (that may have a precedence relation) can also be defined to be simultaneous or distinct (or neither)). Events that are not simultaneous would be constrained by their start events, those that are distinct would be strongly constrained (activities would not overlap).

With a bottom-up development, the activities would come first. If the activities are clearly ordered than so are the mapped events. But if the activities overlap then precedence at the higher band could be considered a derived property. If any observer, operating in the same band as the two events, always observes precedence between the events then the property can be deemed to be true. This approach could be a formal one (for all possible observers) or could be stochastic and provide for a probabilistic assessment of likely consistency.

In general a combination of these top-down and bottom up approach could yield an effective specification scheme that retains the intuitive property of transitivity but provided necessary flexibility and non-determinacy.

5 Band Separation

One of the useful notions of the Framework is:

- *separation* – the property of being able to assume that activities in lower (quicker) bands are instantaneous and the state of higher (slower) bands is unchanging;

This can be defined more precisely as a property of *ordered* bands. Two bands (**A** and **B**) are ordered (**A**>**B**), not just when the granularity of **A** is greater then the granularity of **B**, but in the stronger sense when the granularity of **B** is not greater then the precision of **A**. So, for example, **A** could be the minute band with a precision of 5 seconds, and **B** could be the second band.

An activity of up to 5 seconds in band **B** is instantaneous when viewed from band **A**. Moreover, a state change event in band **A** could last up to 5 seconds (when viewed from band **B**) – and hence any activity of duration less then 5 seconds in band **B** can assume that this state does not change during its 'execution'.

The lecturing example illustrates this property. Whilst giving a lecture one can assume that the curriculum is not changing, and that a 'slide change' event is instantaneous. Whilst planning the curriculum lectures are events. And when designing the laptop's operating system the complex activity that results in a slide change, whilst containing perhaps thousands of events at the hardware level, must meet the timing constraint imposed by the assumption of it being instantaneous when described at the higher band.

It follows from the definition of a mapping, that an event in band **A** can only map to an activity in band **B** if **A** and **B** are ordered: **A**>**B**.

6 Formalization

In this section we summarise the work that has been done to formalise the Timebands framework [40, 42, 38, 37, 36].

To cope with a wide range of time scales, many formal approaches [13, 26] have introduced time granularity, so that system specifications and requirements could be naturally described within the best suitable time granularity. However, they usually transform or project all descriptions into the finest granularity in the end. This results in cumbersome formulae and fails to recognise the distinct role that time is taking in the structuring of the system. For example, it is unnecessary to measure the start of a meeting in a millisecond time scale. In fact, most people are usually tolerant of starting a meeting five minutes early or late. Traditional approaches dealing with time granularity sacrifice the separation of concerns in the analysis of complex real-time systems.

The concept of time granularity has been well defined in the literature [11, 19] and many approaches have focused on time granularity in different areas of computer science, such as temporal databases, data mining, formal specification and so on. Generally speaking, the basic idea of time granularity is to partition a universal time domain into differently-grained granules, and a granularity is a set of indexed granules, any one of which is a set of time instants. The choice for time domain is typically between continuous (or at least dense) and discrete. We focus on developing a natural specification language which is able to describe the behaviour of a real-time system whose components engage in different time scales. In other words, we attempt to embed time granularity in a logical specification language. However, adding time granularity to a formalism may give rise to semantic issues like problems of assigning a proper meaning to statements with different time domains and of switching from one domain to a coarser/finer one. So far, most of work has been focused on embedding time granularity in temporal logic languages. For example, early exploration [13, 26] consists of translation mechanisms that map a formula associated with different time constraints to the finest granularity. They [10] later revise the simple approach by extending the basic logic language with contextual and projection operators, so that the enhanced semantics can express more general and complete properties. Subsequently, more work [12, 15] uses linear time logic to model and reason about time granularity.

For manipulating the unique feature of mapping events into activities, process algebra approaches are potential candidates for formalising the Timebands model. However, there is little work on embedding time granularity in process algebra languages, though there have been many papers [22, 33, 34] on timed process algebra approaches. To formalise the Timebands model, we have proposed a new timed model, named *timed Circus* [40], of *Circus*, which is the combination of Z [35, 44], CSP [17, 30, 33], and the refinement calculus [27] so that it can deal with both data and behavioural aspects of a system. The first denotational semantics of *Circus*, based on Unifying Theories of Programming (UTP) [18], was published in [43]. However it actually describes a *Circus* program as a Z specification in order to use tools like Z/EVES [31] to reason about properties. Later, Woodcock et al. [28] proposed new semantics, also based on UTP, for *Circus* in which each process is described as a reactive design. The so-called reactive designs come from the fact that the new semantics applies the well-defined healthiness conditions of reactive processes to embedding the theory of designs. The new semantics can adopt the sophisticated refinement laws of CSP in the refinement of *Circus* specifications.

We subsequently develop a new timed model [40] which is a compact extension of *Circus*. In other words, *timed Circus* does not inherit Z specification of *Circus*, and also makes use of the latest reactive design semantics in order to mechanically implement the refinement more easily. To some extent, *timed Circus* can also be considered an extension to CSP in the UTP book [18], which is different from the standard CSP models [17, 30, 33]. Therefore, *timed Circus* uses a complete lattice in the implication ordering (or the reverse order of the refinement order), instead of the complete partial order of the standard failures-divergences model of CSP. Prior to our work, Sherif and He [34] proposed a timed model of *Circus* (called *Circus Time*) that also took a subset of *Circus* and created an abstraction function to map the timed model to the original untimed model. However, our timed model uses different semantics for conveniently applying well-defined CSP refinement laws. Moreover, our model extensively explores the role of the reactive design miracle (the top element in the complete lattice) in system specifications, so that we can define brand-new operators to specify some distinctive properties of a system which cannot be easily expressed by other approaches.

6.1 *Timed Circus*

As a compact extension to *Circus*, the syntax of *timed Circus* is close to that of timed CSP except for the deadline and assignment operators. The full syntax is described by the following grammar:

$$\begin{aligned} P ::= & \top_R \mid \bot_R \mid SKIP \mid STOP \mid a \to P \mid P_1 \,;\, P_2 \mid x :=_A e \mid g\&P \mid \\ & P_1 \Box P_2 \mid P_1 \sqcap P_2 \mid P_1 \,|[\, A \,]|\, P_2 \mid P \setminus A \mid WAIT\ d \mid P_1 \rhd \{d\} P_2 \mid \\ & P \blacktriangleright d \mid P_1 \triangle \{a\} P_2 \mid \mu X.P \end{aligned}$$

The definitions of most operators are based on Oliveira et al.'s work [28]. Here, we reconsider it in a time environment.

The miracle $\top_R$ is the top element in the implication ordering, which expresses a process that has not started yet. The bottom element $\bot_R$ is called *CHAOS* which can do absolutely anything. The process *STOP* is deadlocked and its only behaviour is to allow time to elapse. The process *SKIP* simply terminates immediately.

The sequential composition $P_1 \,;\, P_2$ behaves as P_1 until P_1 terminates, and then behaves as P_2. In the meanwhile the final state of P_1 is passed on as the initial state of P_2. The prefix process $a \to P$ is able to execute the event a ($a \in \Sigma$) and then behaves as P. The process $g\&P$ has a boolean expression g which must be satisfied before P starts. The notation $(x :=_A e)$ represents that a process simply assigns the value of an expression e to a process variable x, and then any other variable in the alphabet A remains unchanged.

The process $P_1 \Box P_2$ behaves either like P_1 or P_2, but the first event of which can resolve the choice. Compared with the external choice, the internal choice $P_1 \sqcap P_2$ can also behave either like P_1 or like P_2, but it is out of control of its environment. Both external and internal choices have indexed choices. For example, if I is a finite indexing set such that P_i is defined for each $i \in I$, written as $\Box_{i \in I} P_i$. The indexing external choice is also used to define the input operator. For example, if c is a channel name of type T and v is a particular value, the process $c!v \to P$ outputting v along the channel c is equal to $c.v \to P$. The inputting process $c?x : T \to P(x)$ describes a process that is ready to accept any value x of type T, and it is defined as $\Box_{x \in T} c.x \to P(x)$.

The process $P_1 \,|[\, A \,]|\, P_2$ is the process where all events in the set A must be synchronised, and the events outside A can execute independently. The parallel process terminates only if both P_1 and P_2 terminate, and it becomes divergent after either one of P_1 and P_2 does so. An interleaving of two processes, $P_1 \,|||\, P_2$, executes each part independently and is equivalent to $P_1 \,|[\, \emptyset \,]|\, P_2$. The hiding operator $P \setminus A$ makes the events in the set A become invisible or internal to the process. The process $P_1 \triangle \{a\} P_2$ behaves as P_1, but at any stage before its termination the occurrence of a ($a \notin \alpha P_1$) will interrupt P_1 and pass the program control to P_2. The recursive process $\mu X.P$ behaves like P with every occurrence of the system variable X in P representing a recursive invocation.

The delay process $WAIT\ d$ does nothing except that it allows d time units to pass. The timeout operator $P_1 \rhd \{d\} P_2$ resolves the choice in favour of P_1 if P_1 is able to execute observable (external) events by d time units, otherwise executes P_2. The deadline operator $\blacktriangleright$ is similar to the timeout operator, but it uses the miracle to force that *P must* execute observable events by d. Such a deadline operator is *hard*, that is to say, all deadlines must be met.

Some hard deadline operators [**?**, **?**, **?**] have been proposed both in process algebra approaches and temporal logic approaches. However, besides defining the hard deadline operator, the employment of the reactive design miracle in *timed Circus* provides more powerful and flexible expressiveness in system specifications. For example, in modelling a complex system, it is very convenient to impose a collection of events to happen together. RAISE Specification Language (RSL) [16, 45] has an interlock operator which can prevent the interlocked processes from communicating with other processes until one of them terminates. Of course, the communication can take place between the locked processes if they are able to. Promela/SPIN [20, 21] can define atomic sequences which encapsulate a fragment of code to be executed uninterruptedly and individually. In the interleaving of process executions, no other process can execute statements from the moment that the first statement of an atomic sequence is executed until the last one has completed. Such 'atomic' events can also be easily defined by the deadline operator with well-defined denotational semantics in our *timed Circus* model. For example, setting the value of the deadline as zero can make a process or an event become *instant*. For the sake of convenience, we use the following abbreviations as a shorthand to represent instant events or processes:

$$\begin{aligned} \ddagger P &\mathrel{\widehat=} P \blacktriangleright 0 \\ P_1 \ddagger P_2 &\mathrel{\widehat=} P_1 \,;\, (P_2 \blacktriangleright 0) \\ a \ddagger b &\mathrel{\widehat=} (a \rightarrow \mathit{SKIP}) \ddagger (b \rightarrow \mathit{SKIP}) \end{aligned}$$

Here the instantaneity operator squeezes the 'distance' of events and processes to zero, so that neither a nor b can happen individually.

One potential application of instantaneity is that we can construct *uninterrupted* events in which either all events can happen or none of them can happen individually. For example, we can define a process as follows:

$$P = (a \blacktriangleright 0) \,;\, \mathit{WAIT}\ 1 \,;\, (b \blacktriangleright 0)$$

where a and b are uninterrupted events. That is to say, a can happen only if b can happen one time unit later. Such events are extremely useful for dealing with explicit clock-tick events.

6.2 Semantics of the Timebands Model

In consideration of the nature of the Timebands model, we intend to use *timed Circus* to express its semantics. The newly explored process, the miracle, plays a crucial role in the construction of the Timebands model to link all time bands as a whole. First, we use a lecture example to explain how to view a simple system in the Timebands model. Suppose that one week a lecturer has a lecture which takes two hours and has a five-minute break. To model it, we define three time bands, *Week*, *Hour* and *Minute*, which are given in an increasing finer order and illustrated in Figure 2. In Band *Week*, event *lecture* does not take any time to execute, but it is mapped into activity L with duration in Band *Hour*. Furthermore, event *break* in activity *L* is mapped into another activity *B* in Band *minute*. Thus, instead of mapping all events or activities into the finest band, we use some key events (or signature events) to link and integrate different bands into a whole. Meanwhile, the Timebands model preserves consistency and coordination of the system in the multiple time scales. The Timebands model is developed in a number of stages in this section including time bands, granularity and precision, simultaneous events and durative activities, and mappings between bands.

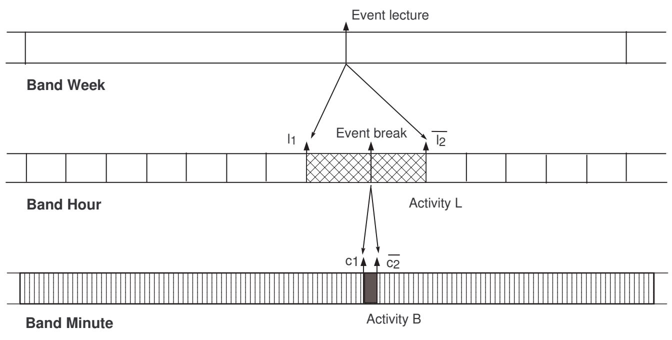

Figure 2: Mapping between different time bands

6.2.1 Time bands

A system in the Timebands model recognises a finite set of distinct time bands, and it always has the highest and the lowest bands that give a temporal system boundary. Each band is defined by a granularity, representing the basic unit of time in that band. This is different from temporal logic approaches which can represent a possibly infinite set of time bands.

The Timebands model adopts continuous time, usually represented by non-negative real numbers. A granule is simply a set of time points and a granularity is a mapping G from integers to granules. One healthiness condition [4] that granularity must satisfy is

$$\boldsymbol{G1} : \forall\, i, j : \mathbb{Z} \mid i < j \wedge G(i) \neq \emptyset \wedge G(j) \neq \emptyset \bullet (\forall\, t : G(i), u : G(j) \bullet t < u)$$

which states that any two granules of a granularity have no overlap and the elements of the granules are ordered the same as their index order. A granule $G(i)$ can be comprised of a single time unit, a set of contiguous units, or even a set of non-contiguous units. For example, the bank holidays for 2009 in England, defined as a collection of several days from different months, can be used as a granule.

Thus, the time bands in the lecture example can be defined as follows:

$$TBI = \{Minute, Hour, Week\}$$
$$Granularity(Hour, Minute) = \{60\}$$
$$Granularity(Week, Hour) = \{7 * 24\}$$

The set *TBI* is a collection of the Timeband identities. The function *Granularity* defines conversion factors of different time bands' granularities. These factors can be multiple. For example, a year may have 365 days or 366 days. In addition, the function *Granularity* also satisfies 'finer than' relations of different time bands. A granularity G is *finer than* a granularity H if for each index i, there exists an index j such that $G(i) \subseteq H(j)$. Conversion factors between two bands must be natural numbers, and therefore time bands are not always comparable. For example, a week band is not comparable to a month band.

6.2.2 Events and precision

Events are instantaneous, but depend on which band they are defined in. For example, an event defined in the week band does not take any time to execute, however it might take several hours in the hour band. Indeed, there are a few relationships between events within a band such as instant events. Instantaneity is the strongest constraint that is used especially to link different time bands via events and activities.

In specifications of a system, events may cause an immediate response. For example, we consider such a requirement like 'when the fridge door opens the light must come on immediately'. It actually means that the two events, *door.open* and *light.on*, occur simultaneously but with an order. That is, the response is within the precision of the band. Precision, representing the measure of accuracy of events within that band, can only be expressed using the granularity of a finer band. Accordingly, two *simultaneous* events must, when viewed from the finer band, be within the precision of the current band. In respect to the finite number of time bands in the model, the finest (lowest) band has no precision. Due to precision, two simultaneous events cannot be exactly distinguished because the 'gap' between them is too small to be considered. Here the small gap also results in tolerance of the behaviours when mapping the two events into the corresponding activities of a finer band. For instance, considering the lecture example with the three bands, precision can be defined as follows:

$$Precision(Week, Hour) = 2$$
$$Precision(Hour, Minute) = 5$$

If event *break* is supposed to happen in the middle of a lecture, the precision of the hour band restricts the maximal duration of a break to be five minutes, otherwise *break* cannot be considered an instantaneous event in the hour band. Also, the precision allows the break to happen five minutes early or late.

Therefore, similar to the definition of instant events, the simultaneous operator is defined as follows:

$$P_1 \overrightarrow{\#} P_2 \mathrel{\widehat{=}} P_1 \,;\, (P_2 \blacktriangleright \rho)$$

where ρ is the precision of that band. Two simultaneous events, e.g., a and b, are expressed as either a is before b or b is before a, but they must occur within the precision.

We also use the following abbreviations to represent simultaneous events:

$$a \overrightarrow{\#} b = (a \to \mathit{SKIP}) \overrightarrow{\#} (b \to \mathit{SKIP})$$
$$a \# b = a \overrightarrow{\#} b \;\Box\; b \overrightarrow{\#} a$$

where $\#$ denotes that a and b are simultaneous, and $\overrightarrow{\#}$ that they are simultaneous but with an order. This abbreviation is applied to all simultaneous events in this paper.

Simultaneity is also a very strong constraint which is similar to instantaneity. That is, either simultaneous events occur together or none of them occurs individually. The difference is that two simultaneous events allow one of them to occur within the precision after the other has occurred, even though such a short delay is too small to be considered in this band. Simultaneous events are the same as instant events if these events are not mapped.

We cannot distinguish two simultaneous events in the current band; however, the interval between simultaneous events will be revealed in the form of precision when mapping these events to corresponding activities in a finer band. As a result, the precision basically plays two roles in a band: one is to measure accuracy of events such as simultaneous events, the other is to restrict the duration of activities. Unfortunately, simultaneity is not transitive, i.e., the fact that a and b are simultaneous and so are b and c, does not imply that a and c are simultaneous. This also elegantly explains that a sequence of consecutively simultaneous pairs or repeatedly fast-moving events can be observing durative behaviours. We might not recognise any pair of them because the interval between them is less than the precision, but the whole duration may take a long time.

6.2.3 Punctual clock-tick events

In modelling of real-time systems, we often employ 'clocks' to aid scheduling and coordination. We represent a default abstract clock in a band by defining each granule as a 'clock-tick' event, which is modelled just like any other event. When necessary, more abstract clocks can be defined by the basic unit of time in the band. For example, the clock called *business days* is placed in the day band; however, it is different from the default day clock.

Timed CSP is unlikely to explicitly represent clock-tick events because it can never guarantee that an event is able to happen precisely at a specific time point. The occurrence of events in timed CSP depends on their environment's interaction even if the timeout operator is applied. However, this situation is entirely changed if we use the deadline operator. For example, we may simply define a punctual clock as follows:

$$C = ((\mathit{tick} \to \mathit{SKIP}) \blacktriangleright 0)\,;\, \mathit{WAIT}\;1\,;\, C$$

where the clock-tick event *tick* must occur precisely every time unit otherwise the punctual clock will not start.

We define clock-tick events for every time band, e.g., the clock-tick events for the lecture example given in previous section can be defined as follows:

$$\begin{aligned}&\mathit{Event}:\mathit{Minute}\ \mathit{mtick}\\&\mathit{Event}:\mathit{Hour}\ \mathit{htick}\\&\mathit{Event}:\mathit{Week}\ \mathit{wtick}\end{aligned}$$

An intuitive way to understand a clock-tick event is that it denotes a start point of a new time unit or the end point of the previous time unit. Therefore, for different clock-tick events in different time bands such as *mtick* and *htick*, we say that the time interval between two *hticks* in the hour band contains 60 *mticks*, rather than that an *htick* can be mapped to an activity in the minute band which includes 60 clock-tick events. That is to say, if a mapping is necessary, a clock-tick event in a higher band is mapped to an activity in a lower band which contains only one clock-tick event.

Punctual clock-tick events provide us with extensive convenience to express clock-related properties. For example, if *tick* is a clock-tick event representing 1:00, $\mathit{tick} \ddagger a$ denotes that a must happen precisely at 1:00 even if we observe it in a finer band; $\mathit{tick}\#a$ means that a must occur at 1:00 too, but a is allowed to happen early or late within the bound of the precision; $\mathit{tick} \to a \to \mathit{SKIP}$ states that a occurs only if its environment provides the offer, and a occurs exactly at 1:00 only if its environment is friendly.

Note that clock-tick events are just ordinary events and they become meaningful only if we let them happen precisely at intervals of one time unit. Therefore, we attach a local timer to any processes during their life cycles. For example, a process with a timer is defined as follows:

$$\begin{aligned}&P_T = ((P\,;\, e \to \mathit{SKIP})\,|[\,\{\mathit{tick}, e\}\,]|\,\mathit{Timer} \bigtriangleup \{e\}\mathit{SKIP}) \setminus \{e\}\\&\mathit{Timer} = \mathit{tick} \to \mathit{Timer}'\\&\mathit{Timer}' = \mathit{WAIT}\ 1 \ddagger (\mathit{tick} \to \mathit{SKIP})\,;\, \mathit{Timer}'\end{aligned}$$

where the event $e\,(e \notin \alpha P)$ is used to stop the timer by the interrupt operator when P terminates, and one time unit in *WAIT* is a local time unit depending on which band the process is defined in. Notice that such a timer does not record the duration of its whole life cycle, while it starts only if the first clock-tick event of the process starts. By comparison with a globally punctual clock in a band, local timers to processes are able to effectively avoid the deadlock caused by asynchronization of clock-tick events[1]. For the sake of convenience, we directly define a process as usual and subsequently its timer is attached automatically.

We usually use a clock-related process to express a very strong constraint that 'something must occur at certain time points'. For example, a process $\mathit{tick} \to a \to \mathit{tick} \to \mathit{SKIP}$ means that a must occur between two clock-tick events. Hence, a

[1]One of the approaches to model-check a timed CSP process is to relate untimed CSP process descriptions to timed ones in the form of *timewise refinement* [32]. This idea is quite powerful, but at the cost of dropping all *WAIT d* terms [29] because of the complexity of synchronising clock-tick events in parallel. However, the mechanism of local timers in our model does not require the synchronisation of all clock-tick events so as to avoid an unnecessary deadlock.

well-defined clock-related process is the one in which all clock-tick events cannot be blocked. For example, a counterexample can be as follows:

$$P = tick \rightarrow tick \rightarrow (WAIT\ 2\,;(tick \rightarrow SKIP))$$

where obviously the third *tick* cannot occur such that the local timer blocks the occurrence of all clock-tick events.

6.2.4 Activities

An activity is a special process with clock-tick events. Activities are detailed explanations of events of higher bands, and hence, to maintain consistency of a system, ‘qualified’ activities must satisfy the following three requirements:

1. An activity must start and also finish with clock-tick events.
2. An activity must have one or more signature events.
3. Duration of an activity must be no longer than the precision of a higher band in which its corresponding event is placed.

Requirement 1 states that an activity should be well placed in the band. If the activity cannot start or finish with clock-tick events, it is supposed to be replaced in a finer time band. For example, an activity may be defined as follows:

$$A = tick\#a_1\,;tick\#\hat{a_2}\,;tick\#a_3$$

which means that the events such as a_1, a_2 and a_3 are simultaneous with clock-tick events, and the duration of the activity is two time units. Note that a_1 may actually occur before the event *tick* in $tick\#a_1$, but we consider that *A* still starts with the clock-tick event since *tick* and a_1 cannot be distinguished in this time band. The duration of *A* is counted from the occurrence of the first *tick*, and not from the start of the activity. That is, the activity may initially wait in silence until the coming of an explicit clock-tick event, and its duration is actually determined by how many clock-tick events it involves.

As Requirement 2, each activity must have one or more signature events, which is not only the major observation of the activity, but also the linking to the corresponding event in a higher band. For example, $\hat{a_2}$ is a signature event in the activity *A* and an overhead line is used to make it different from other ordinary events. An activity can have more than one signature event, which must be linked to the same event of a coarser band and only one of which can happen during the life cycle of the activity. For example, making a drink by a vending machine may have two choices, tea or coffee, which can be described as follows:

$$\begin{aligned} Drink = \;& (tick\#hotwater\,;tick\#milk\,;tick \rightarrow tick\#\widehat{tea}) \\ & \Box\ (tick\#hotwater\,;tick\#milk\,;tick\#\widehat{coffee}) \end{aligned}$$

The duration of an activity should be no longer than the precision of a higher band; otherwise it cannot be considered an event of the higher band. This imperative requirement will be fulfilled when the activity is mapped, since the precision for the activity is not yet decided until the link with the event in the higher band has been established. For example, there are two activities, *A* and *B*, in a day band, but *A* and *B* are linked to two events in a month band and a year band respectively; consequently, their precisions might be different.

When mapping events of a higher band to activities of a lower band, well-defined activities are crucial in maintaining consistency between different time bands. The following three examples are not well-defined activities which violate Requirement 1– 3, respectively:

$$A1 = a \to tick \to \hat{b} \to SKIP$$
$$A2 = tick \to \hat{a} \to tick \to \hat{b} \to tick \to SKIP$$
$$A3 = tick \to \hat{a} \to A3$$

Because an activity is a clock-related process, we can control when the activity will happen by fixing any event of the activity to happen at a specific time. That is, the events in the activity are uninterrupted events. For example, if we impose the signature event $\hat{a_2}$ of the above activity *A* to happen at 10:00, a_1 must then happen at 9:00 and *A* must finish at 11:00. In fact, *A* starts from the beginning of the system; however a_1 can occur only if the other events must occur later.

6.2.5 Mapping between bands

In the components of the model considered so far, all behaviours have been confined to a single band. The essence of the Timebands model is to describe the behaviour of each component of a system in a best suitable time band, and compose the multiple-band behaviours regarding the properties to be verified. To achieve this goal, events in one band may need to be mapped into activities in finer bands.

Activities become useful only when they are linked with events in higher time bands. Processes defined in different time bands have no intersection except for the linking of events and activities. Those links are the one and only channel to integrate all behaviours of the Timebands model. The establishment of the links is achieved by means of imposing the events and the signature events of the activities to be instant events, so that they are constrained to occur together at all time.

The linked pair of an event and an activity can affect each other to decide when they will occur in their own bands. Recall the lecture example illustrated in Figure 2. Activity *B* can be given the following behaviour:

$$\begin{aligned} Event\ {:}Minute\quad & c_1, c_2 \\ Activity\ {:}Minute\quad & B = c_1\#mtick\,; mtick \to mtick \to mtick \to mtick \to \hat{c_2}\#mtick \end{aligned}$$

This activity actually means that students have to take a 5-minute break and any shorter or longer break is not allowed. If we insist that event *break* in the hour band must occur in the middle of the lecture, e.g., around 10:00 (event *break* and the clock-tick event are simultaneous), and then event c_1 in activity *B* can only happen between 9:50 and 10:00, on account of the five-minute precision. That is to say, the signature event $\hat{c_2}$ can happen only between 9:55 and 10:05. We can also set the time when $\hat{c_2}$ in activity

B occurs in the minute band, which alternatively results in the time when event *break* must occur in the hour band. For example, if we say that $\hat{c_2}$ occurs at 9:50, and then *break* in the hour band must occur between 9:00 and 10:00.

To maintain consistency and coordination between different time bands, we simply make events and the signature events of corresponding activities instant. For example, the mapping in the lecture example can be defined as the following processes:

$$Link1 = lecture \ddagger \hat{l_2}$$
$$Link2 = break \ddagger \hat{c_2}$$

And then these processes are synchronised with other processes in the system on all events of the alphabets of $Link1$ and $Link2$.

Finally, we use the lecture example, illustrated in Figure 2, to demonstrate the integration of time bands. Granularity, precision and clock-tick events have been defined in previous sections. In the week band, we specify that a lecture must occur within a week:

$$Event\!:\!Week\ lecture$$
$$LECTURE = wtick \rightarrow lecture \rightarrow wtick \rightarrow SKIP$$

And activity L, expressing a two-hour lecture with a break, is defined in the hour band as follows:

$$Event\!:\!Hour\ l_1, l_2, break$$
$$Activity : Hour\ L = htick\#l_1;\ htick\#break;\ htick\#\hat{l_2}$$

Before events and activities are linked together, processes defined in different time bands have no interaction at all. Thus, the system before mapping is expressed by an interleaving process:

$$S = LECTURE \mid\mid\mid L \mid\mid\mid B$$

And then the integrated system is constructed by linking events *lecture* and *break* with activities L and B respectively:

$$SYS = S \,|[\, \{lecture, l_2\} \,]|\, Link1 \,|[\, \{break, c_2\} \,]|\, Link2$$

In practice, the assumption of maximal progress enables events to occur as soon as possible. For example, the process *LECTURE* in the week band specifies that *lecture* may happen any time within a week, but without a constraint from other processes or bands it always happens at the beginning of the week. With respect to Figure 2[2], if the lecture example starts, *wtick*, *htick* and l_1 will initially start together; *lecture* cannot happen immediately because it is coordinated with $\hat{l_2}$ or the third *htick* in the hour band; c_1 in the minute band cannot happen because it depends on *break* or the second *htick* in the hour band. Subsequently, after one time unit of the hour band, *break* happens; however, $\hat{c_2}$ has occurred five time units (within the precision) of the minute band earlier, since *break* and the second *htick* are simultaneous. Consequently, another hour later, $\hat{l_2}$ and *lecture* happen together.

[2]The clock-tick events are not directly given in this figure, whereas the reader can easily find out where these events should be placed by the description of the system.

6.2.6 Discussion

One of the distinctive features of the Timebands model is its use a mapping between instantaneous events and durative activities to integrate different behaviours described in different time scales into a whole system. The key idea of the mapping is to use instantaneity of the events and the signature events of the corresponding activities, and integrity (uninterrupted events) of the activities to locate right positions for mapped entities. The above two distinct properties are achieved through applying the unique process, the miracle.

The time system of the Timebands model is a combination of implicit time and explicit clock-tick events. Here implicit time, similar to time in timed CSP, is continuous. However, processes themselves do not have read-access to the clock which is rather used in the semantic framework for the analysis and description of processes. A clock-tick event is an observation of a single precise time of the global clock and it can be accessed by any process. Because clock-tick events are punctual, we can specify clock-related events which must occur at specific time points.

Every clock-related process has a local timer (a clock with clock-tick events), which turns out to be able to interfere with the accuracy of its local clock. We do not require that the local clock of a process must be synchronised with the global clock. For example, we let $htick \ddagger a$ to express that a must happen at the beginning of an hour such as 1:00, while we make the signature event (such as $\hat{a}'$) of the corresponding activity simultaneous with *mtick* in the minute band. Thus, $\hat{a}'$ must happen at 1:00 because it is instant to a, but $mtick \# \hat{a}'$ allows its local clock, relative to the clock of the hour band, to quick or slow a little bit within the precision of the minute band. Of course, the local clock of a process can be easily synchronised with the global clock. This property is very useful in modelling the behaviour of a distributed system where components may have asynchronous clocks.

The advantage of the Timebands model is the separation of concerns in dealing with different properties with different time scales. Many properties in the Timebands model involve only few time bands rather than all of time bands. Obviously, apart from a better description of a complex system, proving such properties is more efficient in the Timebands model than the traditional model with a single flat time. In the Section 8, by means of a complicated example, the mine pump, we demonstrate how significantly the Timebands model contributes to describing complex real-time systems with multiple time scales.

7 Discrete and Continuous Behaviour

Many of the properties and primitives defined in the Timebands framework are orientated to discrete notions of time and progress. However this is not a fundamental constraint within the framework. External phenomena that is continuous by nature can be represented by differential equations based on continuous and dense notions of time. If a dynamic entity is associated with a particular band **B**, with granularity G and precision ρ, then two properties need to be defined. These are the maximum *rate-of-change* of the entity (as expressed per G) and the *accuracy* that can be obtained by a reading of the entity expressed as an event (r) in band **B**.

7.1 Possible, Always and Almost Always

An event whist instantaneous in band **B** has a duration of maximum length ρ when mapped to an activity in a finer band. The accuracy of r is defined to be maximum change that can occur within ρ. As the precise instance of a read event cannot be determined a more useful interpretation of a read operation is that it returns a set (or range) of *possible* values for the entity (based on the accuracy attribute). An example from the mine pump case study (used in Section 8) is a requirement that *if the methane reading is high turn pump off* becomes more usefully *if it is possible that the methane is high turn pump off*. Note the pump off event would be defined to *immediately* follow the methane high event. Immediately being instantaneous and order (see earlier discussion).

Whilst important for single events the notion of possible values becomes even more important when two or more simultaneous read operations of different external entities are undertaken. Now the set of possible values of the two entities reflects the accuracies of both [6]. The mine pump example could give rise to the failure being defined as *possible for methane to be high and pump on*.

A state variable may be observed or required to be *always* true for an interval of time, in a specific band. But it is also useful to be able to define a state variable as being true apart from instantaneous point in time where it may be false. We say that a variable is *almost always* true if in any interval of time the only points at which it may be false are 'during' events that are distinct. With this definition it is possible to modify the impossible (or at least meaningless) specification: *this door to remain closed at all times* to *this door to remain almost always closed at all times*. Now an authomatic door controller can open and close the door as long as the complete operation takes less than the precision of the band, and that two such operations are at least two precisions apart. The opposite requirement: *almost never* can be defined similarly.

7.2 Moving Between Discrete and Continuous

Another illustration of the advantage of having both discrete and continuous notions of time within the same framework comes from being able to model behaviour that is discrete in a fine grain band as 'points in time' (i.e., events) in a coarser band. As a result continuous phenomena emerges from discrete. This property of changing granularity was used to move from a discrete view of failures (e.g., a 25ms control cycle either completes in time or a deadline failure occurs) to one in which failures can be represented as a continuous phenomena with a rate, such as 10^{-4}, 'per hour of operation' [8].

The opposite relation comes when discrete events in a coarser band are mapped to activities representing continuous behaviour at the finer band. An example here would be opening and closing of the sluice gate, which could be discrete events in one band and motor driven behaviour with a maximum movement per second in a finer band.

A final issue issue with continuously changing phenomena relates to observing/specifying external entities at more than one time band. For example, a pump may have a defined capability expressed in, say, the minute band: '200 liters per minute'.

At a finer band the maximum rate must be sufficient to deliver this (so in the second band it must be at least 200/60 litres per second). But it could be much more more if, for instance, the pump action was two phased, the maximum rate per second might be 10 litres.

8 Tools and Case Studies

The mine pump example was first proposed by Kramer et al. [23] and later used by Burns and Lister [9] as case study for developing dependable systems. The mine pump system is used to control a pump to pump out the water which is collected in a sump. The mine has two sensors to detect when water is above a high level or below a low level. A pump controller switches the pump on when the water level becomes high and off when it goes below the low level. The system also monitors the level of methane, since a pumping operation during a dangerous methane level will cause explosion. Reading from all sensors, the operations of the mine pump should satisfy the following safety requirements:

1. The pump can be used only when the methane level is safe.
2. The pump must be switched on within an interval since the water level has become high.
3. The pump must be switched off within an interval whenever the methane level becomes dangerous.

In a mine, water and methane come from the environment. We assume that the change of the water level is slow, and the methane level is stable in most of the time but can incidentally change very fast. Therefore, we use two time bands, a minute band and a second band, to describe the slow changing of the water level and the dramatic changing of the methane level respectively. For example, a delay of few seconds may have no influence on the change of the water level, while it could be crucial for switching the pump off when the methane level suddenly becomes dangerous. Granularity and precision between the two bands are defined as follows:

$$\begin{aligned}&\mathit{TBI} = \{\mathit{Minute}, \mathit{Second}\}\\&\mathit{Granularity}(\mathit{Minute}, \mathit{Second}) = \{60\}\\&\mathit{Precision}(\mathit{Minute}, \mathit{Second}) = 5\end{aligned}$$

To simplify the modelling of the mine pump, we abstract the state of the water level as Figure 3 by combining the values of two sensors for detecting the water level. That is, the state of the water level is low until water passes the high level, and it stays high until below the low level. This abstraction is reasonable since it is a practical decision to keep the pump on until the water level becomes low, though sometimes the pump has to be switched off due to the dangerous methane level.

We also assume that each component takes some time to react, e.g., updating values of sensors may take a few seconds, the pump may take some time units to start working

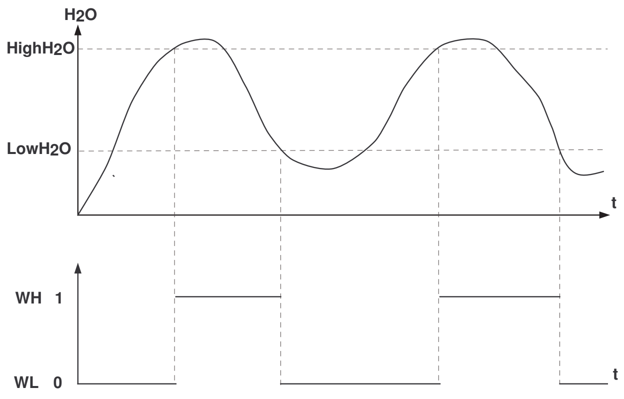

Figure 3: Sample timing diagram for water level

and the sampling frequency also brings delay to update fresh values of states. As a result, reaction time will be considered in the light of how much impact it causes on the safety requirements of the system.

8.1 A pump controller

Depending on the states of the water and methane levels, a pump controller executes actions on the pump. Therefore, in the following implementation defined in the minute band, the system is basically decomposed into two components: one for monitoring the behaviour of water and the other for the behaviour of methane.

$$\begin{aligned} \mathit{Event}:\mathit{Minute}\quad & \mathit{water.high}, \mathit{water.low}, \mathit{pump.on}, \mathit{pump.off}, \\ & \mathit{methane.safe}, \mathit{methane.danger}, \mathit{mtick} \end{aligned}$$

$$\begin{aligned} \mathit{WATER}_{low} = {} & \mathit{water.high} \rightarrow (\mathit{wl} := \mathit{false} \,;\, \mathit{WATER}_{high}) \\ & \Box\, \mathit{pump.off} \rightarrow \mathit{WATER}_{low} \end{aligned} \tag{1}$$

$$\begin{aligned} \mathit{WATER}_{high} = {} & \mathit{water.low} \rightarrow (\mathit{wl} := \mathit{true} \,;\, \mathit{WATER}_{low}) \\ & \Box\, \mathit{pump.on} \rightarrow \mathit{WATER}_{high} \\ & \Box\, \neg\, \mathit{ms}\&\mathit{pump.off} \rightarrow \mathit{WATER}_{high} \end{aligned} \tag{2}$$

$$\begin{aligned} \mathit{METHANE}_{\mathit{safe}} = {} & \mathit{methane.danger} \rightarrow \mathit{METHANE}_{\mathit{danger}} \\ & \Box\ \mathit{pump.on} \rightarrow \mathit{METHANE}_{\mathit{safe}} \\ & \Box\ \mathit{pump.off} \rightarrow \mathit{METHANE}_{\mathit{safe}} \end{aligned} \tag{3}$$

$$\begin{aligned} \mathit{METHANE}_{\mathit{danger}} = {} & \mathit{methane.safe} \rightarrow \mathit{METHANE}_{\mathit{safe}} \\ & \Box\ \mathit{pump.off} \rightarrow \mathit{METHANE}_{\mathit{danger}} \end{aligned} \tag{4}$$

We here remove any time constraint from these components in order to make it become a purely logic judgement for proper operations. For example, process $\mathit{WATER}_{\mathit{low}}$ in (1) states that the water level initially stays at the low level, and it can become high through the event *water.high* and still remain low if executing *pump.off*. In addition, *ms* and *wl* are two state variables to denote the safe methane and low water respectively. In the process $\mathit{WATER}_{\mathit{high}}$, the event *pump.off* can still happen only if the methane level is dangerous.

These components must agree on when the pump is to be switched on or off. For example, before reaching the low water level during the pumping operation, the pump might be switched off due to the dangerous methane level. Afterwards, the pump has to be switched on again until the water level is below the low level.

$$\mathit{CONTROL} = \mathit{WATER}_{\mathit{low}} \,|[\, \{\mathit{pump.on}, \mathit{pump.off}\} \,]|\, \mathit{METHANE}_{\mathit{safe}} \tag{5}$$

Without considering the timing issues, the above implementation *CONTROL* clearly shows that *pump.on* can occur only when the water level is high and the methane level is safe (because *pump.on* is executed from processes $\mathit{WATER}_{\mathit{high}}$ and $\mathit{METHANE}_{\mathit{safe}}$). This satisfies the first safety requirement of the system. However, to make this system closer to reality, we will verify the other two more refined properties, which are going to be modelled in different time bands because of the different behaviours when the water and methane levels are changing.

8.2 Behaviour of water and methane in the minute band

Suppose that the change of the water level is slow and hence its behaviour is captured in the minute band. The methane level is stable for most of the time, but can change very fast; e.g., it can reach the dangerous level in just few seconds. Obviously, such a dramatic change of methane is best described in a finer time band such as the second band. In the following modelling, we will specify the different behaviours of the two components in the two time bands, depending on different scenarios.

For modelling the change of the water and methane levels, we use worst-case execution time to describe the worst situations. As illustrated in Figure 4, the worst situation for water is that the water level has reached the high level but the pump cannot be switched on because the methane level just becomes dangerous. Hence, it is unnecessary to consider any operation when the water level is between the low and high levels if the worst case has satisfied the safety requirements. In practice, we always give a good safety margin to the value of the high level in case the pump cannot be switched

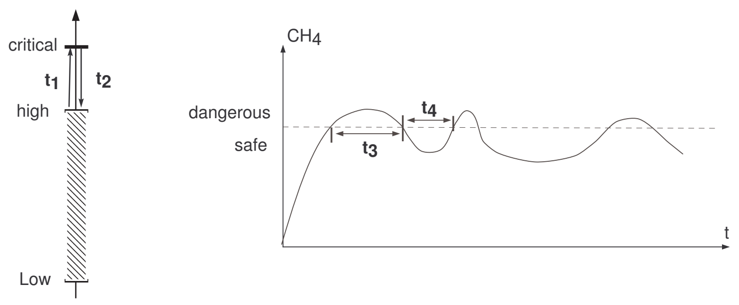

Figure 4: Assumptions on the change of water and methane

on immediately. For example, the pump must be on within t_1 time units after the water level becomes high, otherwise the mine fails. And the pump can take the water level below the high level if it has continuously worked for t_2 time units. If assuming that r_1 and r_2 are the rates of change respectively at which water enters and leaves the mine, we can easily get the equation: $r_1 * t_1 = (r_2 - r1) * t_2$.

Thus, the time constraint of the behaviour of water and the related pump operations in the minute band is modelled as follows:

$$TCW = \mathit{water.high} \rightarrow \mathit{HIGH}(l_1) \tag{6}$$

$$\begin{aligned} \mathit{HIGH}(t_1) = \mathit{IF}\ t_1 == 0\ \mathit{THEN}\ & (\mathit{pump.on} \rightarrow \mathit{ON}(l_1) \,\Box\, \mathit{flooding} \rightarrow \mathit{STOP}) \\ \mathit{ELSE}\ & (\mathit{WAIT}\ 1\,;\, \mathit{HIGH}(t_1 - 1) \\ & \quad \Box\, \mathit{pump.on} \rightarrow \mathit{ON}(l_1 - t_1)) \end{aligned} \tag{7}$$

$$\mathit{ON}(t) = \mathit{OFF}(t * r_1/(r_2 - r_1)) \tag{8}$$

$$\begin{aligned} \mathit{OFF}(t_2) = \mathit{IF}\ t_2 == 0\ & \mathit{THEN}\ \mathit{pump.off} \rightarrow \mathit{water.low} \rightarrow TCW \\ \mathit{ELSE}\ & (\mathit{WAIT}\ 1\,;\, \mathit{OFF}(t_2 - 1) \\ & \quad \Box\, \mathit{pump.off} \rightarrow \mathit{HIGH}(l_1 - t_2 * (r_2 - r_1)/r_1) \end{aligned} \tag{9}$$

where t, t_1 and t_2 are time variables, and l_1, r_1 and r_2 are constants, e.g., l_1 is the maximal value of the bound of t_1. The operator, *IF b THEN P ELSE Q*, is actually a convenient shorthand of a guarded process, $b\&P \,\Box\, \neg\, b\&Q$.

The implementation in (7) states that *pump.on* should happen within some time units if the water level is high. The value of t_1 in $\mathit{HIGH}(t_1)$ is the deadline that *pump.on* must satisfy. If *pump.on* happens before the deadline, the net water level over the high level is recorded and passed to $\mathit{ON}(t)$ in the form of time. Thus, the equation in (8) calculates how long the pump can lower the water level below the high level in line with the value from $\mathit{ON}(t)$. The implementation in (9) denotes that the pump might be switched off before water is below the high level because of the dangerous methane level. If the pump is switched off earlier, the program has to go to *HIGH* again to wait for the occurrence of *pump.on*. However, the maximal interval to make the mine fail obviously becomes shorter or is less than l_1.

Accordingly, the timed behaviour of water and the pump is defined by the following parallel composition:

$$\begin{aligned}&A = \{pump.on, pump.off, water.low, water.high\}\\&TWATER = WATER_{low} \mid[\, A \,]\mid TCW \qquad (10)\end{aligned}$$

The behaviour of methane in the minute band is relatively simple. Under the circumstance of worst-case execution time, we also assume two constants, l_3 and l_4, to be the maximal values of two time variables, t_3 and t_4, as illustrated in Figure 4, to denote the maximal duration of the dangerous methane level and the minimal duration of the safe level respectively.

$$\begin{aligned}TCM &= methane.danger \rightarrow (WAIT\ l_3\ ; \qquad (11)\\&\qquad (methane.safe \blacktriangleright 0)\ ;\ WAIT\ l_4\ ;\ TCM)\\TMETHANE &= METHANE_{safe} \mid[\, \{methane.safe, methane.danger\} \,]\mid TCM \qquad (12)\end{aligned}$$

And then, the system in the minute band can be finally modelled as follows:

$$TCONTROL = TWATER \mid[\, \{pump.on, pump.off\} \,]\mid TMETHANE \qquad (13)$$

Recall the safety properties which are introduced in the beginning of this section. Property 1 can be proved even under the untimed environment. The proof of Property 2 depends on the relationship among those constants. For example, l_1 is obviously greater than l_3, otherwise the mine will fail since the pump cannot be switched on in time. Ideally, l_4 is greater than l_2 or $l_1 * r_1/(r_2 - r_1)$ so that water can be lowered below the high level once the pump is switched on. However, this requirement is too strict to accommodate many patterns of methane's behaviour, e.g., the frequent oscillation around the dangerous level of methane does not satisfy this requirement. Therefore, it is more reasonable to satisfy a looser requirement that l_3/l_4 is less than l_1/l_2 within any interval (whose length should be greater than $l_1 + l_2$).

8.3 Behaviour of methane in the second band

Unfortunately, Property 3 is unsuitable to be verified in the minute band. We know that *pump.off* will happen after *methane.danger* if the pump is on, and this logical order can be nicely proved in the minute band. However, in fact, Property 3 is interpreted as a statement that *methane.danger* and *pump.off* must occur simultaneously. To measure the simultaneous actions of two events, we have to consider the influence of various reaction delays such as transmission delay, reaction delay of the pump and so on, whose behaviours can only be captured in the second band. To model and verify Property 3, we need to explore more details in related events of the minute band.

First of all, we specify precision of the minute band to be 5 seconds, which directly determines the definition of simultaneity and the maximal duration of an activity. The delay of updating the state of water is ignored in the minute band, but it is considered in the second band. We assume the delay to be 2 seconds, and *water.high* and *water.low* are mapped into two activities, WH_s and WL_s, in the second band respectively:

$$Activity : Second \;\; WH_s = stick\#\hat{high};\; stick \rightarrow stick\#whe \tag{14}$$

$$Activity : Second \;\; WL_s = stick\#\hat{low};\; stick \rightarrow stick\#wle \tag{15}$$

where *stick* is a clock-tick event of the second band, *whe* and *wle* denote the end of the two activities respectively, and *low* and *high* are two signature events. In addition, the activities are annotated for convenience.

Moreover, on account of the costing time on updating states and sampling frequency, *methane.danger* is mapped into the following activity:

$$Activity : Second \;\; MD_s = stick\#\hat{danger};\; stick \rightarrow stick \rightarrow stick\#mde \tag{16}$$

And then, with regard to reaction delay, *pump.off* is mapped as well:

$$Activity : Seccond \;\; PF_s = stick\#command\_off;\; (stick \rightarrow stick\#\hat{action\_off}) \tag{17}$$

where the event *action_off* denotes the genuine time when this command takes effect.

Furthermore, we impose a constraint on all of these activities so that none of them can overlap each other because changing states presumably involves some computation:

$$\begin{aligned} ACT_s = \;& (WH_s\,; ACT_s) \,\Box\, (WL_s\,; ACT_s) \\ & \Box\, (MD_s\,; ACT_s) \,\Box\, (PF_s\,; ACT_s) \end{aligned} \tag{18}$$

To verify Property 3 in the second band, the activities in the above implementation are integrated with the minute band by making their signature events instant with the corresponding events of the minute band. For the sake of simplicity, ACT_s is integrated with *CONTROL*, rather than *TCONTROL* with time constraints, because the 'micro' relation of *methane.danger* and *pump.off* is irrelevant with those assumptions on how water and methane change.

$$\begin{aligned} CONTROL_{second} = \;& (CONTROL \;|||\; ACT_s) \\ & |[\,\{water.high, high\}\,]|\, Link3 \\ & |[\,\{water.low, low\}\,]|\, Link4 \\ & |[\,\{methane.danger, danger\}\,]|\, Link5 \\ & |[\,\{pump.off, commmand\_off\}\,]|\, Link6 \end{aligned} \tag{19}$$

Note that these linking processes are just similar to *Link*1 and *Link*2 introduced in Section 6.2.5. Even without the mechanised proof, intuitively, we recognise that Property 3 can be satisfied only if no other event in the minute band occurs between *methane.danger* and *pump.off*, since the total duration of the two events in the second band is 5 seconds. That is, when executing the real program code, the program should directly implement *pump.off* when the methane level is dangerous instead of wasting time on updating the state of water.

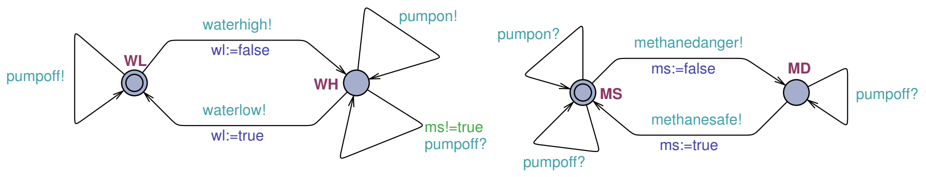

Figure 5: The pump controller without time constraints

8.4 Verification

To prove the three properties of the mine pump example by hand is error-prone since a number of obligations are discharged by obvious and intuitive assumptions where security breaches and system holes are usually hidden. However, establishing the mechanical proof in theorem provers is time-consuming, such as PVS [39] in which the semantics of *timed Circus* has been embedded and ProofPower [46] in which various theories in UTP are mechanised. The model checker FDR [1] is very successful in efficiently verifying both safety and liveness properties of a system modelled in CSP. Therefore, timed CSP specifications can be implemented by FDR if they are translated into untimed ones. However, regardless of its expressiveness, the miracle cannot be expressed in FDR.

Timed automata [2, 25] are powerful in designing real-time models with explicit clock variables, and a number of tools have been proved to be successful like the popular UPPAAL [24]. Timed automata are transition systems consisting of a set of states along with a set of edges to connect these states, and hence it is potential to express the miracle simply as an unstarted state. The idea of using timed automata to implement *timed Circus* or the Timebands model is highly inspired by the work [14] in which they define a set of composable timed automata patterns so that timed CSP can be translated to timed automata. Even if it is possible to represent the miracle in timed automata, the mechanism of the timebands model still involves a massive amount of work. For example, we need to develop a sound operational semantics of *timed Circus* which is usually described as a labelled transition system. We also have to explore a trace-back technique for executing the model, since the fact that a process will not start if the deadline cannot be satisfied means that the process will go back to the unstarted state if the execution cannot go ahead. In the meantime, the observations which have happened during the execution will be erased, and the process just behaves like it has never started. All in all then, the work of fully analysing the Timebands model in timed automata is in progress, and therefore the following verification of the mine pump example in UPPAAL simply provides a flavour to show how it will be possible to prove properties in a model checking approach.

The model checker UPPAAL is based on the theory of timed automata and its modelling language provides expressive features such as urgent edges or locations. The query language of UPPAAL is a subset of TCTL (timed computation tree logic) [**?**]. More explanations of UPPAAL will be given along with the modelling of the mine pump example. First, the process *CONTROL* in (5) is modelled as two timed automata in Figure 5. Locations (or states) of a timed automaton are graphically represented

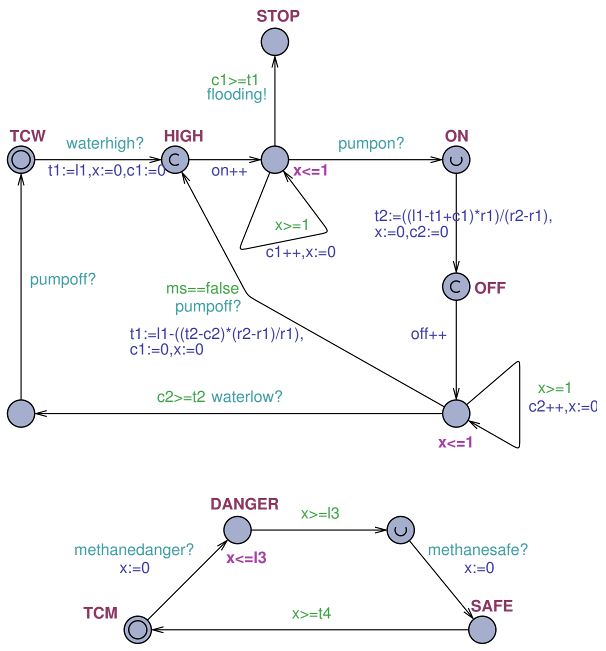

Figure 6: The behaviour of water and methane in the minute band

by circles where the overlapped circle is the initial location. Each location has an invariant which is an expression of conditions, and the program control can stay on this location only if its invariant is satisfied. A transition is a jump from one location to another through an edge which usually consists of three parts: guard, synchronisation and update. For example, illustrated in the left automaton of Figure 5, starting from the location `WH` (*WATER*$_{high}$), event `pumpoff` is synchronised (or fired) with another one in the right automaton only if the methane level is dangerous. As a result, Property 1 holds if the following query is satisfied:

```
A[] METHANE.MS and WATER.WH imply ms==true
```

which means that for all reachable locations, being in the locations `METHANE.MS` and `WATER.WH` implies that `ms==true`. Since `pumpon` can be fired only from the locations `MS` and `WH`, the fact that the methane level is always safe guarantees Property 1.

The behaviour of water and methane in the minute band, *TCW* and *TCM*, are represented by another two automata in Figure 6. Note that `x` in both automata is a local clock that can be reset in the update part of an edge and used in a guard or an invariant. For example, `x` is reset during the transition from location `TCW` to location `HIGH`. Unfortunately, the value of a clock is not allowed to be assigned to any variable in UPPAAL, and that is why we define two integral variables, `c1` and `c2`, to record how long the program control stays on the same location. UPPAAL provides pair-wise

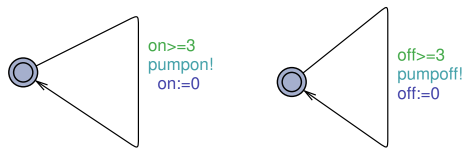

Figure 7: The senders in a multicast synchronisation

synchronisation (one sender and one receiver) via regular channels and broadcast synchronisation (one sender and an arbitrary number of receivers) via broadcast channels. However, a receiver in a broadcast channel can miss the synchronisation if it is not ready yet. Obviously, this is not same as the parallel in timed CSP or *timed Circus*. For example, in the mine pump example, the synchronisation on $pump.on$ and $pump.off$ involving three different processes cannot be directly expressed in UPPAAL. The solution is to use a shared variable (e.g. `on` and `off` in Figure 6) that is increased on the edges leading to a location where those events are ready to happen and is decreased when leaving the location. When the program stays on a location where all events are ready, a sender can be triggered. For example, the senders for $pump.on$ and $pump.off$ are defined as two independent automata in Figure 7.

In addition, urgent (labelled with `U`) and committed (labelled with `C`) locations are used in Figure 6. Time is not allowed to pass when the program is in any of the two locations, but an urgent location can engage in an interleaving. Notice that the approach to calculate the values of t_1 and t_2 in Figure 6 is different from the one in (8) and (9) because a recursive process in the Timebands model is measured by a descending order. To prove Property 2, we simply need to show the automata can never reach location `STOP` or event `flooding` can not be fired if $l_1 > l_3$ and $l_2 < l_4$. Such a query can be expressed as follow:

```
A[] not TCW.STOP
```

which means that it is impossible to reach the location `TCW.STOP`.

To verify Property 3, we should mechanise the miracle so as to express those processes and operators defined by the miracle. However, the embedding of the miracle in UPPAAL is still in progress. Here, regarding the mapping only in the mine pump example, we use an informal scheme to make sure coordination of events and activities in different bands. For example, the process ACT_s, a collection of all activities in the second band, in (18) is described as a timed automaton in Figure 8. The starting of each activity is guarded by a state variable which denotes whether the event in the minute band is ready to happen. For example, the bottom loop (corresponding to MD_s) in Figure 8 states that `danger` can happen only if `ms==true`, and then the automaton waits three time units and finishes the activity with event `mde`. The safe methane level means that the program control is staying on location `MS` as Figure 5, and hence `methanedanger` is ready to occur. The linking processes like *Link* 3–*Link* 6 are expressed as another automaton in Figure 9. The instantaneity of events and the signature events of activities is expressed by committed locations which, however, cannot exactly

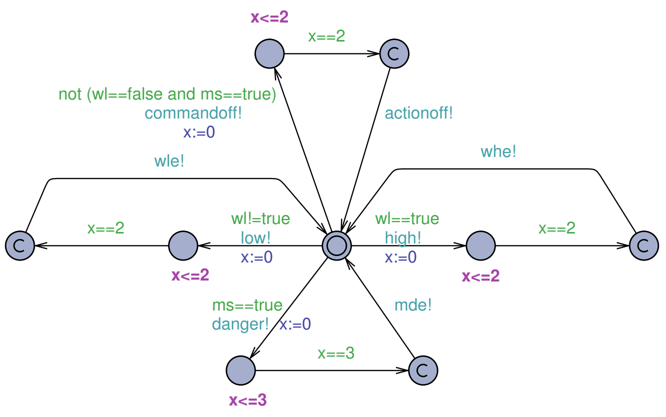

Figure 8: Activities in the second band

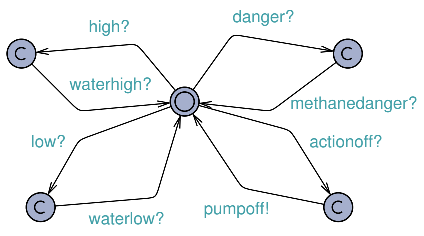

Figure 9: The linking processes in the mapping

describe this property because a committed location just means that time is not allowed to reside and an edge must be fired immediately. If the guard of the edge is not satisfied yet, the automaton is deadlocked.

We add a new location with a guard in the automaton of $METHANE_{safe}$ in order to prove Property 3, as illustrated in Figure 10. The guard on the edge to location `STOP` means if the pump has not been switched off within five time units since the methane level becomes dangerous, the edge can lead to this location. Obviously, we need to prove the following query to be satisfied:

```
A[] not METHANE.STOP
```

The verifier of UPPAAL shows that the above query holds if we impose a constraint to exclude any other events between *methane.danger* and *pump.off*.

9 Conclusions and Future Work

In the work described in this report we have argued that complex real-time systems exhibit behaviour at many different time levels and that a useful aid in describing and

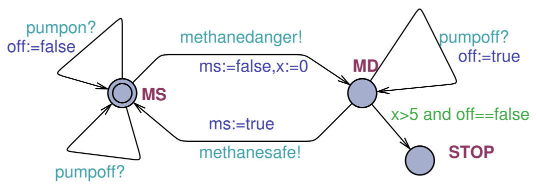

Figure 10: The linking processes in the mapping

specifying such behaviour is to use time bands. Viewing a system as a collection of event and activities within a finite set of bands is an effective means of separating concerns and identifying inconsistencies between different 'layers' of the system. Time bands are not mapped on to a single notion of physical time. Within a system, there will always be a relationship between bands, but the bands need not be tightly synchronised. There is always some level of imprecision between any two adjacent bands.

Since the timeband framework was first defined a number of refinements, additions and corrections have been made. This report defines the current description of the framework. Future work will continue to develop the basic ideas and to evaluate them by further case studies. Another ongoing strand of work is to look to define the framework using the smallest set of primitive notions.

The use of the timeband framework is intended to help develop a comprehensive foundation to the study and development of future systems. Of course an adequately expressive model of time is just one element of such a foundation, but it is perhaps the most important to define if dependable systems are to be engineered.

References

[1] A. W. Roscoe. Model-checking CSP. In *A Classical Mind: essays in Honour of C.A.R. Hoare*, chapter 21. Prentice-Hall, 1994.

[2] Rajeev Alur. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[3] G. Baxter, A. Burns, and K. Tan. Evaluating timebands as a tool for structuring the design of socio-technical systems. In P. Bust, editor, *Contemporary Ergonomics 2007*, pages 55–60. Taylor & Francis, 2007.

[4] Claudio Bettini, Curtis E. Dyreson, William S. Evans, Richard T. Snodgrass, and Xiaoyang Sean Wang. A glossary of time granularity concepts. In *Temporal Databases, Dagstuhl*, pages 406–413, 1997.

[5] A. Burns and G.D. Baxter. Time bands in systems structure. In *Structure for Dependability*, pages 74–90. Springer, 2006.

[6] A. Burns and I.J. Hayes. A timeband framework for modelling real-time systems. *Real-Time Systems Journal*, 45(1–2):106–142, June 2010.

[7] A. Burns, I.J. Hayes, G.Baxter, and C.J. Fidge. Modelling temporal behaviour in complex socio-technical systems. Computer Science Technical Report YCS 390, University of York, 2005.

[8] A. Burns and B. Littlewood. Reasoning about the reliability of multi-version, diverse real-time systems. In *Proceedings of IEEE Real-Time Systems Symposium (RTSS)*, pages 73–81. IEEE Computer Society, December 2010.

[9] Alan Burns and A. M. Lister. A framework for building dependable systems. *The Computer Journal*, 34(2):173–181, 1991.

[10] E. Ciapessoni, E. Corsetti, A. Montanari, and P. San Pietro. Embedding time granularity in a logical specification language for synchronous real-time systems. In *6IWSSD: Selected Papers of the Sixth International Workshop on Software Specification and Design*, pages 141–171, Amsterdam, The Netherlands, The Netherlands, 1993. Elsevier Science Publishers B. V.

[11] James Clifford and Ahobala Rao. A simple, general structure for temporal domains. In *Temporal Aspects in information Systems*, pages 23–30. AFCET, 1987.

[12] Carlo Combi, Massimo Franceschet, and Adriano Peron. Representing and reasoning about temporal granularities. *J. Log. and Comput.*, 14(1):51–77, 2004.

[13] E. Corsetti, A. Montanari, and E. Ratto. Time granularity in logical specifications. In *proceedings of the 6th Italian Conference on Logic Programmming, Pisa, Italy*, 1991.

[14] Jin Song Dong, Ping Hao, Shengchao Qin, Jun Sun, and Wang Yi. Timed automata patterns. *IEEE Trans. Softw. Eng.*, 34:844–859, November 2008.

[15] Massimo Franceschet and Angelo Montanari. Temporalized logics and automata for time granularity. *Theory Pract. Log. Program.*, 4(5-6):621–658, 2004.

[16] The RAISE Language Group. *The RAISE specification language*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[17] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall International, 1985.

[18] C. A. R. Hoare and He Jifeng. *Unifying Theories of Programming*. Prentice-Hall International, 1998.

[19] Jerry Hobbs. Granularity. In *proceedings of the Ninth International Joint Conference on Artificial Intelligence, Los Angeles, California*, pages 432–435, 1985.

[20] Gerard Holzmann. *Spin model checker, the: primer and reference manual*. Addison-Wesley Professional, 2003.

[21] Gerard J. Holzmann. The model checker SPIN. *IEEE Trans. Software Eng.*, 23(5):279–295, 1997.

[22] He Jifeng. From CSP to hybrid systems. pages 171–189, 1994.

[23] J. Kramer, J. Magee, M. Sloman, and A. Lister. CONIC: an integrated approach to distributed computer control systems. *Computers and Digital Techniques, IEE Proceedings E*, 130(1):1–10, January 1983.

[24] Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a Nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1(1–2):134–152, October 1997.

[25] Nancy Lynch and Frits Vaandrager. Action transducers and timed automata. In *Formal Aspects of Computing*, pages 436–455. Springer-Verlag, 1992.

[26] A. Montanari, E. Ratto, E. Corsetti, and A. Morzenti. Embedding time granularity in logical specifications of real-time systems. In *proceedings of the third Euromicro Workshop on Real-Time Systems, Paris, France*, 1991.

[27] Carroll Morgan. *Programming from specifications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1990.

[28] M. Oliveira, A. Cavalcanti, and J. Woodcock. A UTP Semantics for *Circus*. *Formal Aspects of Computing*, 21(1):3 – 32, 2007.

[29] Joël Ouaknine and Steve Schneider. Timed CSP: A Retrospective. *Electr. Notes Theor. Comput. Sci.*, 162:273–276, 2006.

[30] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice-Hall International, 1998.

[31] Mark Saaltink. The Z/EVES system. In *ZUM '97: Proceedings of the 10th International Conference of Z Users on The Z Formal Specification Notation*, pages 72–85, London, UK, 1997. Springer-Verlag.

[32] Steve Schneider. Timewise refinement for communicating processes. *Sci. Comput. Program.*, 28:43–90, January 1997.

[33] Steve A. Schneider. *Concurrent and real-time systems: the CSP approach*. John Wiley & Sons, 1999.

[34] Adnan Sherif and Jifeng He. Towards a time model for *Circus*. In *ICFEM '02: Proceedings of the 4th International Conference on Formal Engineering Methods*, pages 613–624, London, UK, 2002. Springer-Verlag.

[35] J. M. Spivey. *The Z notation: a reference manual*. Prentice Hall International (UK) Ltd., Hertfordshire, UK, 1992.

[36] K. Wei, J. Woodcock, and A. Burns. Formalising the timebands model in timed circus. Technical Report YCS 5xx, University of York, Computer Science Dept., 2010.

[37] K. Wei, J. Woodcock, and A. Burns. Modelling temporal behaviour in complex systems with timebands. In *Conquering Complexity*. 2011.

[38] K. Wei, J. Woodcock, and A. Burns. Timed csp with the miracle. In *In 16th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)*, 2011.

[39] Kun Wei, Jim Woodcock, and Alan Burns. Embedding the timed *Circus* in PVS. Technical Report Available at http://www-users.cs.york.ac.uk/˜kun/, University of York, 2009.

[40] Kun Wei, Jim Woodcock, and Alan Burns. A timed model of *Circus* with the reactive design miracle. In *8th International Conference on Software Engineering and Formal Methods (SEFM)*, pages 315–319, Pisa, Italy, September 2010. IEEE Computer Society.

[41] R. White. *Capturing the temporal properties of complex systems: an evaluation of the timebands approach*. PhD thesis, University of York, Computer Science, York, UK, 2010.

[42] J. Woodcock, M. Oliveira, A. Burns, and K. Wei. Modelling and implementing complex systems with timebands. In *Proc. IEEE Conference on Secure System Integration and Reliability Improvement (SSIRI)*, pages 1–13, 2010.

[43] Jim Woodcock and Ana Cavalcanti. The semantics of *Circus*. In *ZB '02: Proceedings of the 2nd International Conference of B and Z Users on Formal Specification and Development in Z and B*, pages 184–203, London, UK, 2002. Springer-Verlag.

[44] Jim Woodcock and Jim Davies. *Using Z: specification, refinement, and proof.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[45] Xia Yong and Chris George. An operational semantics for timed RAISE. In *FM '99: Proceedings of the Wold Congress on Formal Methods in the Development of Computing Systems-Volume II*, pages 1008–1027, London, UK, 1999. Springer-Verlag.

[46] F. Zeyda and A. Cavalcanti. Mechanical reasoning about families of UTP theories. In *Electronic Notes in Theoretical Computer Science*, pages 240:239–257, July 2009.