# Chapter 13
# Modelling Temporal Behaviour in Complex Systems with Timebands

**Kun Wei, Jim Woodcock, and Alan Burns**

## 13.1 Introduction

Complex real-time systems exhibit dynamic behaviours on many different time levels. For example, circuits have nanosecond speeds for computation in a component, whereas slower functional units may take seconds to achieve their goals; moreover, the involvement of human activities related to calendar units such as days, weeks, months and even years may take more time. To cope with a wide range of time scales, many approaches [10, 23] 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.

To overcome the above weakness when traditional approaches model dynamic temporal behaviours of a system, Burns and Hayes [5] propose a timebands model in which a system is decomposed to reveal different behaviours in different time bands. Apart from defining time bands by granularities, a key aspect of the timebands framework is that *events* are considered to be instantaneous in a band, and then in a finer band they can be mapped into *activities* that have duration. For example, to express a statement that *every month we have a meeting which lasts one hour*, we model the meeting as an instantaneous event in a month band and subsequently map it into an activity in an hour band. This clearly allows dynamic temporal behaviours to be partitioned, but not to be isolated from each other. The mapping

K. Wei (✉)
Department of Computer Science, University of York, York, UK
e-mail: kun@cs.york.ac.uk

M. Hinchey, L. Coyle (eds.), *Conquering Complexity*,
DOI 10.1007/978-1-4471-2297-5_13, 

between different bands leads to more distinct features. For example, precision is introduced to represent the measure of accuracy of events within a band; accordingly, events are *simultaneous* only if, when viewed from a finer band, their corresponding activities are within the precision. Activities may overlap even though their corresponding events in a coarser band are well-ordered. As a result, a formal model of the timebands framework is needed to allow consistency to be asserted between different temporal descriptions that are specified in different time bands.

The concept of time granularity has been well defined in the literature [8, 16] 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. General 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 (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 work has been focused on embedding time granularity in temporal logic languages. For example, early exploration [10, 23] consists of translation mechanisms that map a formula associated with different time constraints to the finest granularity. They [7] 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 [9, 12] 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 [19, 28, 29] on timed process algebra approaches. To formalise the timebands model, we have proposed a new timed model of CSP, called timed CSP with the miracle ($TCSP_M$), which is an extension to timed CSP [28] but whose semantics is based on *Unifying Theories of Programming* (UTP) [15]. This new model uses a complete lattice with respect to the implication order (or the reverse order of the refinement order), which is rather different from previous models such as the complete partial order of CSP [14, 25, 28]. The semantics of the timebands model is built upon $TCSP_M$, fully applying the miracle (the top element of the complete lattice) to express those brand-new features such as simultaneous events and mappings. In this chapter, we use a mine pump example to show how naturally to verify different temporal properties using the timebands model at different time scales. The idea and informal description of the timebands framework has been given in [5], and the formal semantics of the framework is developed in this chapter.

The chapter is structured as follows. We begin with a brief introduction of $TCSP_M$ in Sect. 13.2. Section 13.3 presents how to use the new timed model to

formalise the timebands model and how to formally express these distinct features. Then, by means of a rather complex example, we demonstrate how significantly the timebands model contributes to describing a complex real-time system with multiple time scales in Sect. 13.4. Section 13.5 concludes this chapter.

## 13.2 Timed CSP with the Miracle

Recently, we have proposed a new timed model [32] of *Circus* [26, 33] which is a combination of CSP, Z [34] and the refinement calculus so as to define both data and behavioural aspects of a system. In fact, our timed *Circus* is a compact extension of *Circus* in that it inherits only the CSP part in order to reduce the difficulty of implementing *Circus* programs in practice. Although it does not have the same capability of handling data as the original *Circus* language does, our timed *Circus* preserves local variables for each process that still contains a considerable power to express the change of states. To formalise the timebands model, we further simplify timed *Circus* to $TCSP_M$ by adopting discrete time.

Simply speaking, $TCSP_M$ can also be considered an extension to Schneider's timed CSP [28], but its UTP-style semantics uses a complete lattice in the implication ordering which is different from the complete partial order of timed CSP. With the application of the miracle (the top element of the model), $TCSP_M$ turns out to be able to express some surprising behaviours which, moreover, cannot be described in timed CSP. Additionally, $TCSP_M$ violates some axioms of the standard failures-divergences model of CSP, e.g., traces are not prefix closed any more.

In UTP, Hoare and He use the alphabetised relational calculus to give a denotational semantics that can explain a wide variety of programming paradigms. Hence, the alphabet of a process $P$ in $TCSP_M$ consists of *undashed* variables $(a, b, \ldots)$ and *dashed* variables $(a', x', \ldots)$. The former, written as $in\alpha\, P$, stands for initial observations, and the latter as $out\alpha\, P$ for intermediate or final observations. The relation is then called *homogeneous* if $out\alpha\, P = in\alpha\, P'$, where $in\alpha\, P'$ is simply obtained by putting a dash on all the variables of $in\alpha\, P$. Thus, an observation in $TCSP_M$ is a tuple consisting of $tr$, $ref$, $ok$, $wait$, $t$, $v$ and their dashed counterparts, in which $tr$ and $tr'$ are timed traces, $ref$ and $ref'$ are refusals, $ok$ is a boolean variable expressing whether a process has started or not ($ok'$ whether the process has terminated or not), $wait'$ denotes whether the process is in an intermediate state, $t$ is the starting time of the observation ($t'$ is the finishing time), and $v$ and $v'$ denote a set of local variables of the process.

A timed trace is a sequence of timed events which are pairs drawn from $\mathbb{Z}^+ \times \Sigma$,[1] e.g., $\langle (1, pump.on), (3, pump.off) \rangle$ is a timed trace. A refusal is simply a set of events, other than a set of time events in timed CSP, since other variables can assist in representing enough information of when those events are refused. The $ok$ and $wait$ observations (and their dashed variables) describe whether a process is started

[1] $\Sigma$ denotes the universal set of events.

(or finished) in a stable state. If $ok'$ is false, the process diverges. If $ok'$ is true, the state of the process depends on the value of $wait'$. If $wait'$ is true, the process is in an intermediate state, otherwise it successfully terminates. Similarly, the values of undashed variables represent the states of the process's predecessor.

Except for the deadline and assignment operators, the syntax of $TCSP_M$ is similar to the one of timed CSP, as described by the following grammar:

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

### 13.2.1 Primitive Processes

The miracle $\top_R$ is the top element in the implication ordering, however it cannot be executed since it expresses that a process has not started yet. Of course, an unstarted process satisfies any requirement. The bottom element $\bot_R$ is called *Abort* 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.

### 13.2.2 Sequential

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 $a \rightarrow P$ is able to execute the event $a$ ($a \in \Sigma$) and then behaves as $P$. This process can also be represented by a composition of a simple prefix and $P$ itself, written as $(a \rightarrow SKIP) ; P$. 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 any other variable in the alphabet $A$ remains unchanged. In practice, we often use a shorthand for the assignment operator. For example, $P(x + 1)$ is actually defined as $(x := x + 1 ; P)$.

### 13.2.3 Choice

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 this 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 $c!v \rightarrow P$ outputting $v$ along the channel $c$ is equal to $c.v \rightarrow P$. The inputting process $c?x : T \rightarrow P(x)$ describes a process that is ready to accept any value $x$ of type $T$, and it is defined as $\Box_{x \in T} c.x \rightarrow P(x)$.

### 13.2.4 Parallel

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 $P_1 \,|[\, \emptyset \,]|\, P_2$.

### 13.2.5 Abstraction and Recursion

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 $a$ 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. For example, to express a simple recursive process $P = a \rightarrow P$, we have a monotonic function $F$, a variable $X$, and an equation $F(X) = a \rightarrow X$; and then $P$ is actually represented by $\mu X.F(X)$ which stands for the least fixed point of the above equation.

### 13.2.6 Timed Operators

The delay process $WAIT\ d$ does nothing except that it allows $d$ time units to pass. The timeout operator $P_1 \triangleright \{d\} P_2$ resolves the choice in favour of $P_1$ if $P_1$ is able to execute observable (external) events by $d$ time units, otherwise executes $P_2$. This operator is defined by the combination of the external choice and hiding operators:

$$P_1 \triangleright \{d\} P_2 = ((P_1 \,;\, e \rightarrow SKIP) \,\Box\, (WAIT\, d \,;\, e \rightarrow P_2)) \setminus \{e\}$$

which uses the event $e$ to resolve the external choice, if no external event happens in $P_1$ by $d$ or $P_1$ does nothing but terminates before $d$. Also, $e$ is not included in the alphabet of $P_1$ and $P_2$.

The deadline operator ▶ is similar to the timeout operator, but it uses the miracle to force that $P$ *must* execute observable events by $d$:

$$P \blacktriangleright d = (((P\,;e_1 \rightarrow SKIP) \,\Box\, (WAIT\ d\,;e_2 \rightarrow STOP)) \setminus \{e_2\}$$
$$\Box\, WAIT\ d\,;\top_R) \setminus \{e_1\}$$

where the role of $e_1$ ($e_1 \notin \alpha P$) is to resolve both external choices when $P$ quietly terminates before $d$, and the event $e_2$ ($e_2 \notin \alpha P$) is used to resolve the first external choice if $P$ does nothing when $d$ is due. Of course, the miracle ($\top_R$) forces $P$ to execute external events, otherwise the whole process will behave like the miracle. More detailed explanation of the deadline operator will be found in Sect. 13.2.9 after some algebraic laws are introduced. This is really a very strong requirement in which there is no alternative but to meet the deadline, otherwise $P$ will never start.

Note that our deadline operator is different from the one defined in timed CSP which, in fact, indicates that a process becomes deadlocked if events cannot occur by $d$. The deadline operator in $TCSP_M$ insists that the process will not start at all if the deadline is missed. In other words, events in $P$ must occur by $d$, or the process behaves as $\top_R$.

### 13.2.7 Refinement

Suppose that $P_1$ and $P_2$ have the alphabet $A$ of variables. If every observation that satisfies $P_1$ also satisfies $P_2$, it is expressed by $\forall v : A \bullet P_1 \Rightarrow P_2$, or $[P_1 \Rightarrow P_2]$. Because the refinement order is the reverse order of implication, it can also be written as $P_2 \sqsubseteq P_1$. The miracle is an unstarted process so that its observation obviously satisfies any other process in the model, i.e., $[\top_R \Rightarrow P]$ or $P \sqsubseteq \top_R$.

### 13.2.8 The Difference from Timed CSP

Although $TCSP_M$ inherits assumptions of timed CSP such as maximal parallelism and maximal progress, the introduction of the miracle makes $TCSP_M$ different from timed CSP in many aspects. The miracle itself is a very 'strange' process since it can never be executed in practice. However, it is very useful as a mathematical abstraction in reasoning about properties of a system. The semantics of $\top_R$ in $TCSP_M$ is defined as follows:

$$\top_R = (tr \leq tr' \wedge t \leq t' \wedge \neg ok) \vee (wait \wedge ok' \wedge \mathcal{I})$$

where $\mathcal{I}$ is called relational identity which simply means that all dashed variables in the alphabet are equivalent to correspondingly undashed variables. The observation of the miracle consists of two parts: the left part of the disjunction states that, since

*ok* is false, its predecessor diverges and the miracle is in an unstable state; the second one states that the miracle is waiting for its predecessor's termination (e.g., *wait* is true) but in a stable state (e.g., $ok'$ is true). However, in both cases, the miracle has not started yet.

The miracle gives rise to some very strange processes, each of which violates one of axioms of the standard CSP failures-divergences model. For example, we combine the miracle with a simple prefix, and then get the following miraculous process:

$$a \rightarrow \top_R \mathrel{\widehat{=}} R(true \vdash tr' = tr \wedge a \notin ref' \wedge wait' \wedge v' = v) \tag{13.1}$$

Let us concentrate on the expression after the symbol $\vdash$, which describes the behaviour if a process starts from a stable state. The reader who is interested in $R, \vdash$ and proof is referred to [31, 32]. The process (13.1) states that, if the process starts stably, then it will wait for interaction with its environment ($wait'$ is true), but never actually perform any event ($tr' = tr$) even if the event $a$ has been offered ($a \notin ref'$). This process violates an axiom of the CSP failures-divergences model [25, 28],

$$F3.\ (s, X) \in F \wedge \exists a \in Y \bullet s \frown \langle a \rangle \notin traces_{\perp}(P) \Rightarrow (s, X \cup Y) \in F$$

saying if at a state an event is not in the refusal set then the process is willing to execute the event.

Another strange process is that the external choice of the miracle with a simple prefix:

$$(a \rightarrow SKIP) \,\Box\, \top_R = R(true \vdash \neg wait' \wedge tr' = tr \frown \langle (t', a) \rangle \wedge v' = v) \tag{13.2}$$

In an untimed model this process performs the event $a$ and terminates immediately. There is no state in which the process is waiting for the environment to offer $a$. It simply occurs instantly; in other words, no empty trace exists for such a process. Obviously, it violates another important axiom of the standard failures-divergences model of CSP where traces are prefix closed. In our timed model, this process reveals more interesting features. Because there is no constraint on timing in (13.2), the event $a$ will occur when the environment is willing to interact with it. However, there is still no state between the start of the process and the occurrence of $a$, or the time before the occurrence of $a$ has become invisible.

### *13.2.9 Distinct Features*

The combination of the miracle and other operators can further assist us in understanding the role of the miracle. In fact, the key role of the miracle in a process is that the program control should never meet the miracle if the process has started.

This idea can be applies to intuitively get the following laws[2]:

$$L1.\ \top_R ; P = \top_R$$

$$L2.\ SKIP ; \top_R = \top_R$$

$$L3.\ STOP \,\Box\, \top_R = \top_R$$

$$L4.\ SKIP \,\Box\, \top_R = SKIP$$

$$L5.\ P \sqcap \top_R = P$$

$$L6.\ P \,|[\,\{A\}\,]|\, \top_R = \top_R$$

For example, the left part in $L2$ should not start and therefore behaves as $\top_R$ since *SKIP* allows the program control to meet the miracle immediately if the process starts. Similarly, the process in $L4$ must behave as *SKIP* to discard the miracle. The process in $L6$ states that the parallel of the miracle with any process is the miracle, because all processes engaged in the parallel must start together, however, the miracle cannot start so that the whole parallel cannot too.

#### 13.2.9.1 Deadline

The deadline operator in $TCSP_M$ is different from the one defined in timed CSP. It can be used to specify a property that *something must occur*, rather than that something should occur otherwise the process is deadlocked. For example, $(a \rightarrow SKIP) \blacktriangleright 1$ means that $a$ must occur within one time unit, or the process will not start if the deadline cannot be satisfied. An easy way to understand this property is to note that the process will backtrack to the unstarted state if $a$ cannot happen within the deadline.

We can further clarify how the deadline operator works from its definition. For example, there are three cases in which $P \blacktriangleright d$ will behave: the first one is that $P$ executes external events before $d$, another two are that $P$ does nothing by $d$ and $P$ does nothing but terminates by $d$ respectively. The first and third cases are straightforward to implement in the definition of $\blacktriangleright$ since the external events and $e_1$ will resolve both external choices. We focus on the second case and use a simple example to prove its correctness.

$$\begin{aligned}(WAIT\ \ 2) \blacktriangleright 1 &= (((WAIT\ 2 ; e_1 \rightarrow SKIP) \,\Box\, (WAIT\ 1 ; e_2 \rightarrow STOP)) \setminus \{e_2\} \\ &\quad \Box\, WAIT\ 1 ; \top_R) \setminus \{e_1\} \\ &= WAIT\ 1 ; (((WAIT\ 1 ; e_1 \rightarrow SKIP) \,\Box\, (e_2 \rightarrow STOP)) \setminus \{e_2\} \\ &\quad \Box\, \top_R) \setminus \{e_1\} \\ &= WAIT\ 1 ; ((e_2 \rightarrow STOP) \setminus e_2) \,\Box\, \top_R\end{aligned}$$

[2]These laws have been formally proved and the reader is referred to [31].

$$= WAIT\ 1\ ;\ (STOP \,\Box\, \top_R)$$
$$= WAIT\ 1\ ;\ \top_R$$

The result is very interesting. As our previous conclusion that a program control should never meet the miracle during any execution, $WAIT\ 1\ ;\ \top_R$ actually means that the process will behave like the miracle unless it can be interrupted before one time unit. As a result, the above example proves that if a process cannot execute external events by the deadline, it behaves as the miracle. In other words, if the process can then it must do so.

#### 13.2.9.2 Atomic Events

In modelling a complex system, it is very convenient to impose a collection of events to happen together. For example, RAISE Specification Language (RSL) [13, 35] 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 [17, 18] 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. Unfortunately, to our best knowledge, neither of the two operators has denotational semantics probably because of the insufficient capability of current languages to express the property that something must occur.

Such 'atomic' events can also be easily defined by the deadline operator with well-defined denotational semantics. 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:

$$\ddagger P \mathrel{\widehat{=}} P \blacktriangleright 0$$
$$P_1 \ddagger P_2 \mathrel{\widehat{=}} P_1\ ;\ (P_2 \blacktriangleright 0)$$
$$a \ddagger b \mathrel{\widehat{=}} (a \rightarrow SKIP) \ddagger (b \rightarrow SKIP)$$

Here the instantaneity operator squeezes the 'distance' of events and processes to zero. In addition, none of instant events can happen individually. Moreover, we can define *uninterrupted* events by means of the instantaneity operator. For example, $(a \rightarrow WAIT\ \ 1) \ddagger (b \rightarrow SKIP)$ means that $a$ can happen only if $b$ can even if there is one time unit delay between them. Such events are extremely useful for dealing with explicit clock-tick events in Sect. 13.3.3.

### 13.2.10 Discussion

$TCSP_M$ is a discrete-time version of timed *Circus*, which is also considered an extension to timed CSP. Its denotational semantics is based on UTP by embedding the theory of designs in the theory of reactive systems. More detailed introduction to its semantics can be found in [31, 32]. To prove the correctness and consistency of the model, we have done a shallow embedding [30] of the semantics of our timed *Circus* in the theorem prover PVS. The behaviours of our strange processes have been proved by hand and also by PVS. The ongoing work is focusing on the operational semantics of the timed model and the development of efficient tool support.

## 13.3 Semantics of the Timebands Model

In consideration of the nature of the timebands model, we intend to use $TCSP_M$ 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 Fig. 13.1. 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.

### 13.3.1 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 discrete time, usually represented by non-negative integers. A granule is simply a set of time points and a granularity is a mapping $G$ from integers to a granule. One healthiness condition [4] that granularity must satisfy is

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

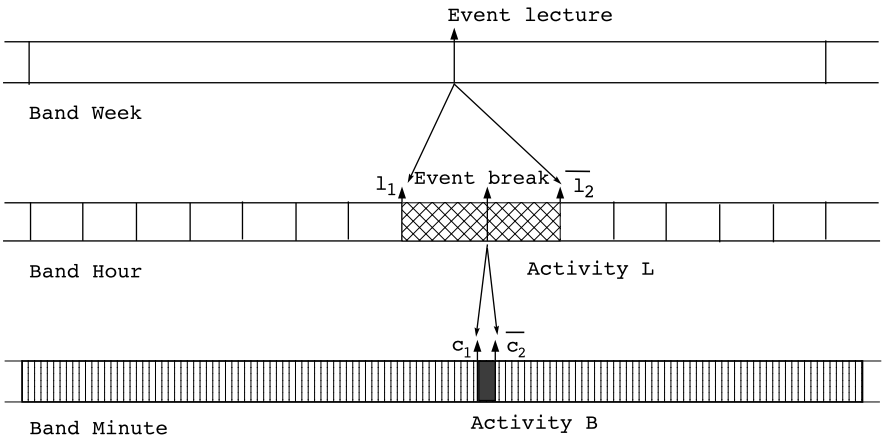



**Fig. 13.1** Mapping between different time bands

which states that any two granules of a granularity have no overlap and the elements of 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.

### *13.3.2 Events and Precision*

Events are instantaneous, but depend on which band they are defined. 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 defined in Sect. 13.2.9. Instantaneity is the strongest constraint that is used especially to link different time bands via events and activities.

In specification 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 finer bands. Accordingly, two *simultaneous* events must, when viewed from a 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 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 $\rho$ 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:

$$a \overrightarrow{\#} b = (a \rightarrow SKIP) \overrightarrow{\#} (b \rightarrow SKIP)$$

$$a \# b = a \overrightarrow{\#} b \,\Box\, b \overrightarrow{\#} a$$

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 chapter.

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.

### *13.3.3 Punctual Clock-Tick Event*

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. However these clock-tick events are forced to happen at same intervals by the deadline operator. 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:

$$C = ((tick \rightarrow SKIP) \blacktriangleright 0)\,;\, WAIT\ 1\,;\, C$$

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:

$$Event : Minute\ mtick$$

$$Event : Hour\ htick$$

$$Event : Week\ wtick$$

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, $tick \ddagger a$ denotes that $a$ must happen precisely at 1:00 even if we observe it in a finer band; $tick \# a$ means that $a$ must occur at 1:00 too, but $a$ is allowed to happen early or late within the bound of the precision; $tick \rightarrow a \rightarrow SKIP$ 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:

$$P_T = ((P \,;\, e \rightarrow SKIP) \,|[\,\{tick, e\}\,]|\, Timer \triangle \{e\}SKIP) \setminus \{e\}$$

$$Timer = tick \rightarrow Timer'$$

$$Timer' = WAIT\ 1 \ddagger (tick \rightarrow SKIP) \,;\, Timer'$$

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 synchronisation of clock-tick events.[3] 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 $tick \rightarrow a \rightarrow tick \rightarrow SKIP$ means that $a$ must occur between two clock-tick events. Hence, a well-defined clock-related process is 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.

[3] One of the approaches to model-check a timed CSP process is to translate it into an untimed CSP one in the form of *timewise refinement* [27]. This idea is quite powerful, but at the cost of dropping all *WAIT* $d$ terms [24] 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.

### *13.3.4 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:

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

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, $\overline{a_2}$ 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:

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

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 Requirements 1–3, respectively:

$$A1 = a \rightarrow tick \rightarrow \overline{b} \rightarrow SKIP$$

$$A2 = tick \rightarrow \overline{a} \rightarrow tick \rightarrow \overline{b} \rightarrow tick \rightarrow SKIP$$

$$A3 = tick \rightarrow \overline{a} \rightarrow A3$$

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, as introduced in Sect. 13.2.9. For example, if we impose the signature event $\overline{a_2}$ 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 $a_1$, very similar to the event of (13.1) in Sect. 13.2.8, can occur only if the other event must occur later.

### *13.3.5 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 Fig. 13.1. Activity $B$ can be given the following behaviour:

$$Event : Minute\ c_1, c_2$$

$$Activity : Minute\ B = c_1 \# mtick ; mtick \rightarrow mtick \rightarrow mtick \rightarrow mtick \rightarrow \overline{c_2} \# mtick$$

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 $\overline{c_2}$ can happen only between 9:55 and 10:05. We can also set the time when $\overline{c_2}$ 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 $\overline{c_2}$ 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:

$$Link1 = lecture \ddagger \overline{l_2}$$

$$Link2 = break \ddagger \overline{c_2}$$

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 Fig. 13.1, 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:

$$Event : Hour\ l_1, l_2, break$$

$$Activity : Hour\ L = htick\#l_1; htick\#break; htick\#\overline{l_2}$$

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:

$$S = LECTURE \,|||\, L \,|||\, B$$

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 anytime within a week, but without a constraint from other processes or bands it always happens at the beginning of the week. With respect to Fig. 13.1,[4] if the lecture example starts, *wtick*, *htick* and $l_1$ will initially start

[4]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.

together; *lecture* cannot happen immediately because it is coordinated with $\overline{l_2}$ 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, $\overline{c_2}$ 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, $\overline{l_2}$ and *lecture* happen together.

### *13.3.6 Discussion*

The revolution of the timebands model is to 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 a global clock whose granularity is the basic time unit of the finest time band. 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 $htick‡a$ to express that $a$ must happen at the beginning of an hour such as 1:00, while we make the signature event (such as $\overline{a'}$) of the corresponding activity simultaneous with *mtick* in the minute band. Thus, $\overline{a'}$ must happen at 1:00 because it is instant to $a$, but $mtick\#\overline{a'}$ 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 following section, 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.

## 13.4 Case Study

The mine pump example was first proposed by Kramer et al. [20] and later used by Burns and Lister [6] 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:

$$TBI = \{Minute, Second\}$$

$$Granularity(Minute, Second) = \{60\}$$

$$Precision(Minute, Second) = 5$$

To simplify the modelling of the mine pump, we abstract the state of the water level as Fig. 13.2 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 takes a few seconds, the pump may take some time units to start working 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.

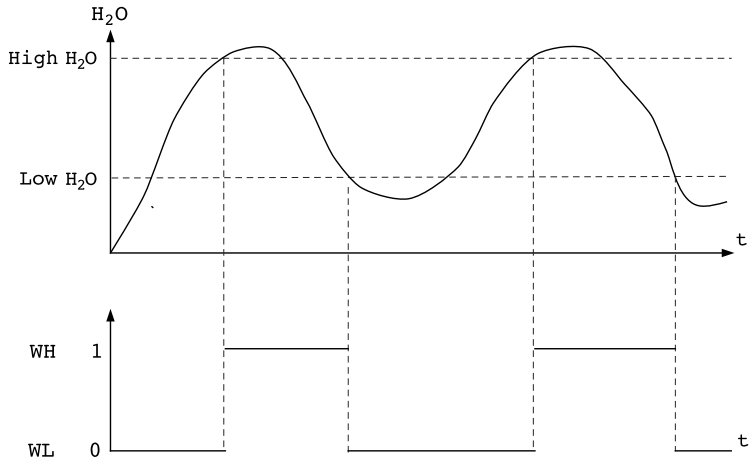



**Fig. 13.2** Sample timing diagram for water level

### *13.4.1 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.

$$
\begin{aligned}
&Event : Minute \quad water.high, water.low, pump.on, pump.off, \\
&\qquad methane.safe, methane.danger, mtick \\
&WATER_{low} = water.high \rightarrow (wl := false\,;\, WATER_{high}) \\
&\qquad \Box\, pump.off \rightarrow WATER_{low} \qquad (13.3) \\
&WATER_{high} = water.low \rightarrow (wl := true\,;\, WATER_{low}) \\
&\qquad \Box\, pump.on \rightarrow WATER_{high} \\
&\qquad \Box\, \neg ms \& pump.off \rightarrow WATER_{high} \qquad (13.4) \\
&METHANE_{safe} = methane.danger \rightarrow METHANE_{danger} \\
&\qquad \Box\, pump.on \rightarrow METHANE_{safe} \\
&\qquad \Box\, pump.off \rightarrow METHANE_{safe} \qquad (13.5) \\
&METHANE_{danger} = methane.safe \rightarrow METHANE_{safe} \\
&\qquad \Box\, pump.off \rightarrow METHANE_{danger} \qquad (13.6)
\end{aligned}
$$

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

$WATER_{low}$ in (13.3) states that the water level initially stays at the low level, and it can become high through 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 $WATER_{high}$, 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.

$$CONTROL = WATER_{low} \,|[\, \{pump.on, pump.off\} \,]|\, METHANE_{safe} \tag{13.7}$$

Without considering the timing issues, the above implementation *CONTROL* clearly shows that event *pump.on* can occur only when the water level is high and the methane level is safe (because *pump.on* is executed from processes $WATER_{high}$ and $METHANE_{safe}$). 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.

### *13.4.2 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 Fig. 13.3, 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 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: $r_1 * t_1 = (r_2 - r1) * t_2$.

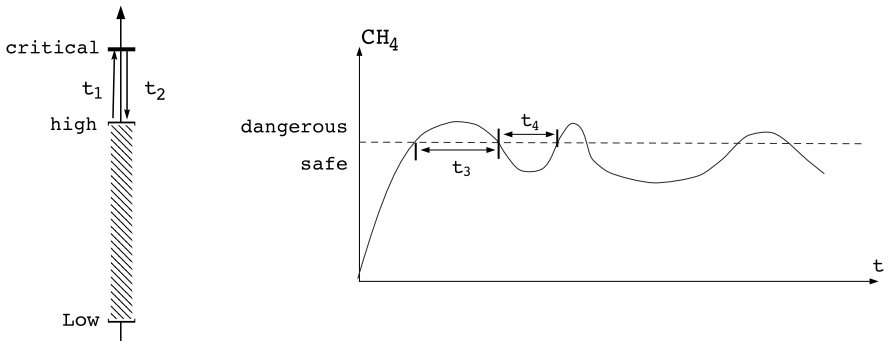



**Fig. 13.3** Assumptions on the change of water and methane

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

$$TCW = water.high \rightarrow HIGH(l_1) \tag{13.8}$$

$$\begin{aligned} HIGH(t_1) = IF\ t_1 == 0\ THEN\ (&pump.on \rightarrow ON(l_1) \\ &\Box\, flooding \rightarrow STOP) \\ ELSE\ (&WAIT\ 1\,;\, HIGH(t_1 - 1_1) \\ &\Box\, pump.on \rightarrow ON(l_1 - t_1)) \end{aligned} \tag{13.9}$$

$$ON(t) = OFF(t * r_1/(r_2 - r_1)) \tag{13.10}$$

$$\begin{aligned} OFF(t_2) = IF\ t_2 == 0\ THEN\ &pump.off \rightarrow water.low \rightarrow TCW \\ ELSE\ (&WAIT\ 1\,;\, OFF(t_2 - 1_1) \\ &\Box\, pump.off \rightarrow HIGH(l_1 - t_2 * (r_2 - r_1)/r_1) \end{aligned} \tag{13.11}$$

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 (13.9) states that *pump.on* should happen within some time units if the water level is high. The value of $t_1$ in $HIGH(t_1)$ 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 $ON(t)$ in the form of time. Thus, the equation in (13.10) calculates how long the pump can lower the water level below the high level in line with the value from $ON(t)$. The implementation in (13.11) 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:

$$A = \{pump.on, pump.off, water.low, water.high\}$$
$$TWATER = WATER_{low} \,|[\, A \,]|\, TCW \tag{13.12}$$

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 Fig. 13.3, to denote the maximal duration of the dangerous methane level and the minimal duration of the safe level respectively.

$$TCM = methane.danger$$
$$\rightarrow WAIT\ l_3\,;\ (methane.safe \blacktriangleright 0)\,;\ WAIT\ l_4\,;\ TCM \tag{13.13}$$
$$TMETHANE = METHANE_{safe} \,|[\, \{methane.safe, methane.danger\} \,]|\, TCM \tag{13.14}$$

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

$$TCONTROL = TWATER \,|[\, \{pump.on, pump.off\} \,]|\, TMETHANE \tag{13.15}$$

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 $l_1 * r_1/(r_2 - r_2)$ 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 $l_1 + l_2$).

### *13.4.3 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:

$$Activity : Second\ WH_s = stick\#\overline{high}; stick \rightarrow stick\#whe \tag{13.16}$$

$$Activity : Second\ WL_s = stick\#\overline{low}; stick \rightarrow stick\#wle \tag{13.17}$$

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:

$$Activity : Second\ MD_s = stick\#\overline{danger}; stick \rightarrow stick \rightarrow stick\#mde \tag{13.18}$$

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

$$\begin{aligned} Activity : Second\ PF_s = {} & stick\#command\_off; stick \\ & \rightarrow stick\#\overline{action\_off} \end{aligned} \tag{13.19}$$

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:

$$\begin{aligned} ACT_s = {} & (WH_s ; ACT_s) \,\Box\, (WL_s ; ACT_s) \\ & \Box\, (MD_s ; ACT_s) \,\Box\, (PF_s ; ACT_s) \end{aligned} \tag{13.20}$$

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.

$$\begin{aligned} CONTROL_{second} = {} & (CONTROL \,|||\, ACT_s) \\ & |[\,\{water.high, high\}\,]|\, Link3 \\ & |[\,\{water.low, low\}\,]|\, Link4 \\ & |[\,\{methane.danger, danger\}\,]|\, Link5 \\ & |[\,\{pump.off, commmand\_off\}\,]|\, Link6 \end{aligned} \tag{13.21}$$

Note that these linking processes are just similar to *Link*1 and *Link*2 introduced in Sect. 13.3.5. 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.

### *13.4.4 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 [30] in which the semantics of $TCSP_M$ has been embedded and ProofPower [36] 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, 22] 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 [21]. 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 $TCSP_M$ or the timebands model is highly inspired by the work [11] 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 $TCSP_M$ 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

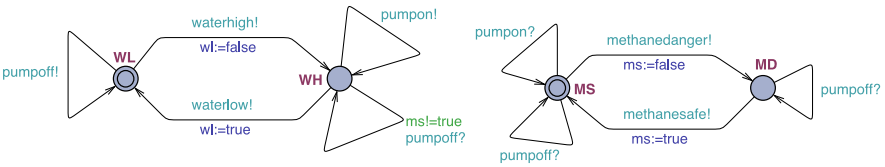



**Fig. 13.4** The pump controller without time constraints

logic) [3]. More explanations of UPPAAL will be given along with the modelling of the mine pump example. First, the process *CONTROL* in (13.7) is modelled as two timed automata in Fig. 13.4. Locations (or states) of a timed automaton are graphically represented 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 Fig. 13.4, 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 Fig. 13.5. 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 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 $TCSP_M$. 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 Fig. 13.5) 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 Fig. 13.6.

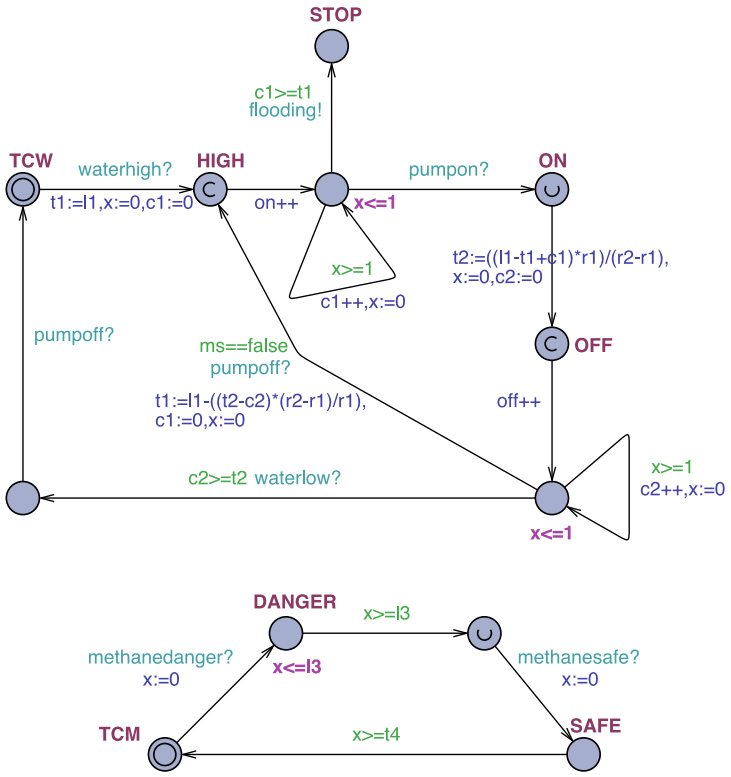
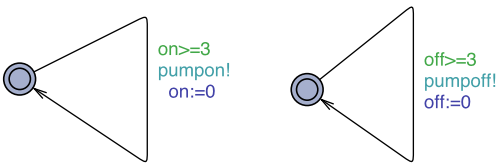



**Fig. 13.5** The behaviour of water and methane in the minute band

**Fig. 13.6** The senders in a multicast synchronisation




In addition, urgent (labelled with `U`) and committed (labelled with `C`) locations are used in Fig. 13.5. 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 Fig. 13.5 is different from the one in (13.10) and (13.11) 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`.

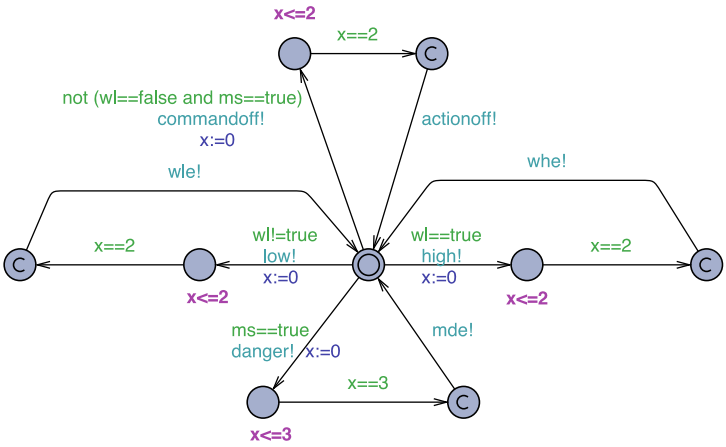



**Fig. 13.7** Activities in the second band

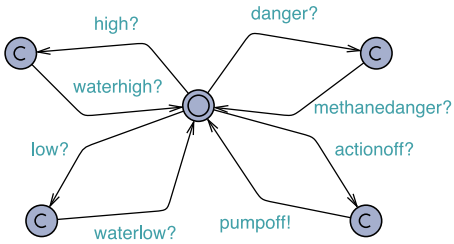



**Fig. 13.8** The linking processes in the mapping

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 (13.20) is described as a timed automaton in Fig. 13.7. 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 Fig. 13.7 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 Fig. 13.4, and hence `methanedanger` is ready to occur. The linking processes like *Link* 3–*Link* 6 are expressed as another automaton in Fig. 13.8. The instantaneity of events and the signature events of activities is expressed by committed locations which, however, cannot exactly 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 Fig. 13.9. The guard on the edge to location

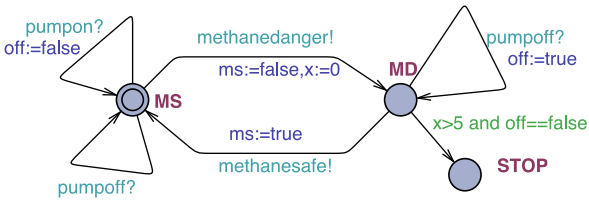



**Fig. 13.9** The linking processes in the mapping

`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*.

## 13.5 Conclusion

In this chapter we have formalised the timebands model using a new timed model ($TCSP_M$) and shown how significantly the model contributes to describing dynamic behaviours of complex real-time systems at many different time scales. Viewing a system as a collection of behaviours within a finite set of bands and integrating these behaviours through linking events and corresponding activities are a natural and effective approach to separate concerns and identify inconsistencies between different time bands of the system. We have also demonstrated the potential to use timed automata to implement the timebands model. Of course, it is still a long way to go for fully mechanising the timebands model. In future work we will apply the timebands framework to the analysis of more complex systems such as socio-technical systems. We believe that the modelling with a time-based hierarchy is able to help develop a comprehensive foundation to dependable systems.

**Acknowledgements** We would like to thank Ana Cavalcanti, Leo Freitas, Andrew Butterfield and Pawel Gancarski for discussions on the role of reactive miracles in programming logic, and thank Cliff Jones and Ian Hayes for discussion on the timebands model and possible approaches to formalisation. This work was partially supported by INDEED project funded by EPSRC:Grant EP/E001297/1.

## References

1. Roscoe A.W.: Model-checking CSP. In: A Classical Mind: Essays in Honour of C.A.R. Hoare. Prentice Hall, New York (1994). Chap. 21
2. Alur, R.: A theory of timed automata. Theor. Comput. Sci. **126**, 183–235 (1994)

3. Alur, R., Courcoubetis, C., Dill, D.L.: Model-checking for real-time systems. In: LICS, pp. 414–425 (1990)
4. Bettini, C., Dyreson, C.E., Evans, W.S., Snodgrass, R.T., Wang, X.S.: A glossary of time granularity concepts. In: Temporal Databases, Dagstuhl, pp. 406–413 (1997)
5. Burns, A., Hayes, I.J.: A timeband framework for modelling real-time systems. Real-Time Syst. **45**(1–2), 106–142 (2010)
6. Burns, A., Lister, A.M.: A framework for building dependable systems. Comput. J. **34**(2), 173–181 (1991)
7. Ciapessoni, E., Corsetti, E., Montanari, A., San Pietro, P.: 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, pp. 141–171. Elsevier, Amsterdam (1993)
8. Clifford, J., Rao, A.: A simple, general structure for temporal domains. In: Temporal Aspects in Information Systems, pp. 23–30. AFCET, Paris (1987)
9. Combi, C., Franceschet, M., Peron, A.: Representing and reasoning about temporal granularities. J. Log. Comput. **14**(1), 51–77 (2004)
10. Corsetti, E., Montanari, A., Ratto, E.: Time granularity in logical specifications. In: Proceedings of the 6th Italian Conference on Logic Programming, Pisa, Italy (1991)
11. Dong, J.S., Hao, P., Qin, S., Sun, J., Yi, W.: Timed automata patterns. IEEE Trans. Softw. Eng. **34**, 844–859 (2008)
12. Franceschet, M., Montanari, A.: Temporalized logics and automata for time granularity. Theory Pract. Log. Program. **4**(5–6), 621–658 (2004)
13. Group, T.R.L.: The RAISE Specification Language. Prentice Hall, Upper Saddle River (1993)
14. Hoare, C.A.R.: Communicating Sequential Processes. Prentice Hall International, Englewood Cliffs (1985)
15. Hoare, C.A.R., Jifeng, H.: Unifying Theories of Programming. Prentice Hall International, Englewood Cliffs (1998)
16. Hobbs, J.: Granularity. In: Proceedings of the Ninth International Joint Conference on Artificial Intelligence, Los Angeles, California, pp. 432–435 (1985)
17. Holzmann, G.: Spin Model Checker, the Primer and Reference Manual. Addison-Wesley, Reading (2003)
18. Holzmann, G.J.: The model checker SPIN. IEEE Trans. Softw. Eng. **23**(5), 279–295 (1997)
19. Jifeng, H.: From CSP to hybrid systems. In: A Classical Mind: Essays in Honour of C.A.R. Hoare, pp. 171–189. Prentice Hall International, Hertfordshire (1994)
20. Kramer, J., Magee, J., Sloman, M., Lister, A.: CONIC: an integrated approach to distributed computer control systems. IEE Proc. E, Comput. Digit. Tech. **130**(1), 1–10 (1983)
21. Cardellini, V., Casalicchio, E., Grassi, V., Lo Presti, F., Mirandola, R.: Uppaal—a tool suite for automatic verification of real-time systems. In: Hybrid Systems III. LNCS, vol. 1066, pp. 232–243. Springer, Berlin (1995)
22. Lynch, N., Vaandrager, F.: Action transducers and timed automata. In: Formal Aspects of Computing, pp. 436–455. Springer, Berlin (1992)
23. Montanari, A., Ratto, E., Corsetti, E., Morzenti, A.: Embedding time granularity in logical specifications of real-time systems. In: Proceedings of the Third Euromicro Workshop on Real-Time Systems, Paris, France (1991)
24. Ouaknine, J., Schneider, S.: Timed CSP: a retrospective. Electron. Notes Theor. Comput. Sci. **162**, 273–276 (2006)
25. Roscoe, A.W.: The Theory and Practice of Concurrency. Prentice Hall International, Englewood Cliffs (1998)
26. Sampaio, A., Woodcock, J., Cavalcanti, A.: Refinement in *Circus*. In: FME '02, pp. 451–470. Springer, London (2002)
27. Schneider, S.: Timewise refinement for communicating processes. Sci. Comput. Program. **28**, 43–90 (1997)
28. Schneider, S.A.: Concurrent and Real-Time Systems: The CSP Approach. Wiley, New York (1999)

29. Sherif, A., He, J.: Towards a time model for *Circus*. In: ICFEM '02: Proceedings of the 4th International Conference on Formal Engineering Methods, pp. 613–624. Springer, London (2002)
30. Wei, K., Woodcock, J., Burns, A.: Embedding the timed *Circus* in PVS. Technical report. Available at http://www-users.cs.york.ac.uk/~176kun/, University of York (2009)
31. Wei, K., Woodcock, J., Burns, A.: Formalising the timebands model in timed *Circus*. Technical report. Available at http://www-users.cs.york.ac.uk/~kun/, University of York (2010)
32. Wei, K., Woodcock, J., Burns, A.: A timed model of *Circus* with the reactive design miracle. In: 8th International Conference on Software Engineering and Formal Methods (SEFM), pp. 315–319, IEEE Comput. Soc., Pisa (2010).
33. Woodcock, J., Cavalcanti, A.: 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, pp. 184–203. Springer, London (2002)
34. Woodcock, J., Davies, J.: Using Z: Specification, Refinement, and Proof. Prentice Hall, Upper Saddle River (1996)
35. Yong, X., George, C.: 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, pp. 1008–1027. Springer, London (1999)
36. Zeyda, F., Cavalcanti, A.: Mechanical reasoning about families of UTP theories. Electron. Notes Theor. Comput. Sci. **240**, 239–257 (2009)