Article:
Burns, Alan orcid.org/0000-0001-5621-8816, Hayes, Ian and Jones, Cliff (2019) Deriving specifications of control programs for cyber physical systems. Computer journal. ISSN 1460-2067

https://doi.org/10.1093/comjnl/bxz019

Deriving specifications of control programs for cyber physical systems

ALAN BURNS[1], IAN J. HAYES[2] AND CLIFF B. JONES[3]

[1]*Department of Computer Science, University of York, YO10 5GH, UK*
[2]*School of Information Tech. and Electrical Eng, The University of Queensland, 4072, Australia.*
[3]*School of Computing Science, Newcastle University, NE1 7RU, UK*
Email: alan.burns@york.ac.uk

Cyber Physical Systems (CPS) exist in a physical environment and comprise both physical components and a control program. Physical components are inherently liable to failure and yet an overall CPS is required to operate safely, reliably and cost effectively. This paper proposes a framework for deriving the specification of the software control component of a CPS from an understanding of the behaviour required of the overall system in its physical environment. The two key elements of this framework are (i) an extension to the use of rely/guarantee conditions to allow specifications to be obtained systematically from requirements (as expressed in terms of the required behaviour in the environment) and nested assumptions (about the physical components of the CPS); and (ii) the use of time bands to record the temporal properties required of the CPS at a number of different granularities. The key contribution is in combining these ideas; using time bands overcomes a significant drawback in earlier work. The paper also addresses the means by which the reliability of a CPS can be addressed by challenging each rely condition in the derived specification and, where appropriate, improve robustness and/or define weaker guarantees that can be delivered with respect to the corresponding weaker rely conditions.

Keywords: Cyber-physical systems; real-time systems; time bands; rely-guarantee; concurrency; embedded systems

Received August 8th, 2018; revised December 7th, 2018

1. INTRODUCTION

The construction of large socio-technical real-time systems, such as those found in cyber-physical applications, presents a number of significant challenges, both technical (see for example [1]) and organisational. The complexity of such systems makes all stages of their development (requirements analysis, specification, design, implementation, deployment and maintenance/evolution) subject to failure and costly re-working. This paper presents a framework that makes it possible to relate the specification of the overall system to that of a suitable control program. The objective being to suggest a way to produce resilient systems that degrade gracefully when physical failures occur.[4]

Typically Embedded Systems (ES) and Cyber Physical Systems (CPS) are comprised of a control program linked (by sensors and actuators) to physical components that can both sense and bring about changes in the physical world. The overall system requirements (the "client's view") are about what should happen in the physical world. Many authors (e.g. [2]) have argued that this is therefore a sensible place to ground a specification.

Given that there are known techniques for developing programs to satisfy formal specifications, an obvious challenge is that of arriving at a formal specification for software control components of an ES or CPS. This is a challenge addressed by many authors (see Section 5 on related work).

The idea of starting from an understanding of what should happen in the physical world was followed in previous work but encountered a serious complication that made it impracticable to tackle larger applications. The key contribution of this paper is to link to the "time band" framework to overcome the complication.

Earlier publications by the current authors [3, 4] argued that:

- for an ES, it is normally easier to establish the requirements of the overall system (than to specify the control system in isolation) by relating the specification to the physical phenomena;
- in general, it is impractical to model the physical components of the ES completely and it is far more realistic to record assumptions about their behaviour;
- both the overall specification and the assumptions must be reviewed with the client;
- further, to analyse safety aspects one requires an overall system specification;
- notation such as rely/guarantee conditions can be used

[4]There are of course other important considerations with CPS such as performance and security. Whilst recognising their importance, these issues are not addressed in the current paper.

to record both the wider system behaviour and the assumptions about the physical sub-components of the ES;
- it is possible to prove that, under the assumption that the physical components behave according to their specifications, the control software will ensure the overall system requirements are met.

The cited publications however ran into a serious problem which was the description of the temporal behaviour. The timing of everything from the overall system through the physical sub-components of the ES down to the behaviour of the control program has to be described, and if the description is given on a single time index (as in [3, 4]) the descriptions become opaque. Moreover, it is alarmingly easy to make mistakes by not realising the consequences of formulae.[5]

The key development in this paper is the combination of the ideas above with the "Time Band" concept introduced in [5] (and expanded upon in this paper). The approach here argues that systems tend to exhibit temporal behaviour at a number of different granularities and trying to present a specification on a single time axis becomes unworkable.

This paper brings together a collection of ideas (introduced in Section 2) to provide an overall framework for specifying the required behaviour of the software components of embedded systems and, more generally, cyber physical systems.

The method of presentation employed in this paper is to develop an illustrative example (in Section 3) of one use of the framework.[6] This illustrates that the complications in the earlier papers [3, 4] –that resulted from expressing assumptions and commitments on a single time axis– can be neatly resolved by using time bands [5]. The central role of this paper is to show that this unique combination of research ideas offers a realistic way of deriving the specification of a control system from the overall requirements of a CPS.

Section 4 examines fault tolerance; related work is covered in Section 5 and conclusions drawn in Section 6.

The notation used in the current paper is intended to be illustrative: it is not all formally defined nor is it claimed that it would cover all examples. This paper aims to set out the general approach — later work will formalise one or more sets of detailed notation and proof obligations. Our experience with working on industrial applications (e.g. [6, 7]) indicates that it is necessary to fit in with specific notations used.

2. CONCEPTS UNDERLYING THE FRAMEWORK

This section briefly introduces the theories and modelling techniques that are integrated into our proposed framework. Citations are provided to detailed descriptions; illustrations of their use are given in the case study. Our focus in this paper is concepts rather than formal description which will be provided in later publications.

2.1. Rely/Guarantee conditions

The idea of specifying discrete, isolated, components with pre conditions (predicates of a single state) and post conditions (relations between initial and final states) is familiar from program development methods such as VDM [8], Z [9], B [10] and Event-B [11]. Rely and guarantee conditions were put forward in [12, 13, 14] as a way of specifying and developing shared-variable concurrent programs.[7]

The basic Rely/Guarantee idea is simple: just as post conditions abstract from any algorithm to achieve the transition from initial to final state, guarantee conditions record a relation –in advance of designing the algorithm– that defines the maximum interference that the eventual code can inflict on its environment. The analogue of a pre condition (which records assumptions that the developer can make about the states in which the component will be initiated), is a rely condition that records a relation that expresses the interference that the eventual code must tolerate. It is important to bear in mind that pre and rely conditions are assumptions that a developer is invited to make; in contrast, guarantee and post conditions are obligations on the code to be created.

Although the rely/guarantee idea originated in the context of the design of concurrent programs, it became clear that the environment of a component could include physical components. Some history of this evolution is mentioned in [3, 4]. Developing hybrid applications such as braking systems for cars brings in the need to use rely conditions on continuously varying values. Assumptions about, for example, the maximum rate of change of speed provide a basis for deciding sampling rates on sensors. As well as capturing assumptions about the physical world in which the CPS will operate, rely conditions can also be used to document assumptions about the physical sub-components of a CPS.

Figure 1 illustrates, for a CPS that is constructed from software (control) and the physical hardware, the top-level use of rely (R) and guarantee conditions (G). Behaviourally, the requirement is that the guarantee condition needs to be satisfied (only) as long as the rely condition is respected. Stating this negatively, if the environment makes a transition that is not in accord with the rely condition, the developed code is free from further obligations (just as an implementation is unconstrained if invoked in a starting state that does not satisfy its pre condition).

Unsurprisingly, the proof obligations concerning the development of concurrent components (that are specified with rely/guarantee conditions) are more complicated than

[5] A specific instance of this in drafts of [4] was that a version of a temporal formula combined the ratio of time that the gate should be open with details about the time to activate the gate mechanism. On close inspection, it was realised that the single formula incorrectly forced copying of the pattern in the first hour to all subsequent periods. Using time bands, the ratio of open:close times is expressed separately from details of gate traversal timing and each band has a separate precision.

[6] Some other examples are mentioned in Section 6.

[7] A significant reworking [15, 16, 17] of the rely/guarantee approach provides a more algebraic view of the various assumptions and constraints by recasting them in the refinement calculus style [18, 19].

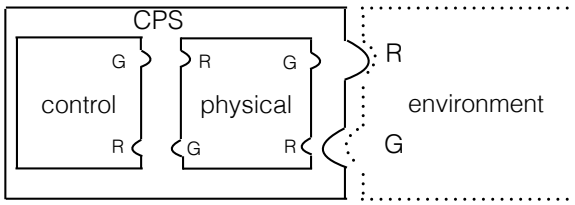

FIGURE 1. Top-Level Description of CPS

those for sequential programs using Hoare-triples [20]. The proof obligations for concurrency include the matching of rely and guarantee conditions mentioned above as well as laws for the distribution of such conditions over sequential and concurrent program compositions. The key point is that discharging the proof obligations establishes that a design that uses a particular programming construct will, if all of the sub-components satisfy their specifications, satisfy the overall specification. This gives rise to the crucial property that development is compositional in the sense that the specification of a component determines everything that governs its functional acceptance; when developing a component from its specification, neither the specification of the larger system, nor those of sibling components need to be taken into consideration.

A single R/G specification can have internal structure so that in the event of some aspects of the rely condition not being true, other aspects that are still satisfied will ensure that some behaviours are guaranteed. Indeed resilience can be represented by hierarchically related R/Gs – strong rely conditions guarantee full functionality, weaker rely conditions guarantee less (perhaps only the safety-critical parts), even weaker rely conditions guarantee only safe fail-stop behaviour. This is discussed further in Section 4.

2.2. HJJ and Refinement

An ES consists of a software control system and physical system components; the connections between the former and the latter are sensors and actuators. The overall behaviour of the ES clearly depends on both aspects. It is argued above that the overall system is easier to specify than the controller itself; what is proposed here is that the specification of the software control system can be derived from the overall system specification. To check this formally, it must be possible to reason about the behaviour of the physical components but, in general, building a complete model of such physical components is unrealistic and it is far easier to document (reviewable) assumptions about their behaviour.

The test then is, given the specification of the control system and the assumptions about the physical components, can it be shown that the overall system specification follows. The 'and' in the previous sentence behaves like an addition; what is actually needed is something more like subtraction. Given the specification of the overall system, one records assumptions about the physical components and derives a specification for the control system that will make the above addition hold.

The observation that rely/guarantee thinking can be applied to systems that interact with the physical world led to an approach that is sometimes referred to briefly by the initial letter of the family names of its three initiators (Hayes, Jackson and Jones): the 'HJJ' approach [3, 4]:

- the phenomena of interest with CPS often vary continuously
- it is nearly always easier to obtain agreement on the behaviour expected of an overall system than it is to start by specifying the control system
- this follows from the fact that assumptions have to be made about both the behaviour of components that are external to the system and physical components within the system

The HJJ approach advocates that the (continuous) behaviour of the overall system is described first in terms of phenomena that can be observed in the physical world; assumptions about those things outside the control system are then recorded (these include assumptions about external components and physical components of the CPS). The specification of the control system is that, if all of the assumptions are fulfilled, the behaviour of the cyber system is such that the required overall system behaviour is achieved. It is clearly mandatory that, as well as agreeing the overall specification with the customer/client, the validity of the assumptions about the physical components are agreed.

In summary, HJJ is the process of moving from requirements that are expressed in terms of state in the environment, to the specification of the control system which can only include input/output data (and internal variables); in making this transition, assumptions are recorded using rely and guarantee conditions. Refinement can then be applied to translate the software specification into executable code.

Experience with the approach described in [3, 4] was positive but had a major drawback that descriptions of various requirements on a single time axis became confusing and unintuitive. Resolving this with the help of time bands is precisely the contribution of the current paper.

2.3. Time Bands

The above technologies adequately address the functional properties of the ES or CPS, but do not cater for the important temporal aspects of these systems. The motivation for the timebands framework comes from a number of observation about complex CPS [5, 21, 22, 23, 24]:

- the dynamics of a system (how quickly things change) are central to understanding its behaviour
- systems can be best understood by distinguishing different granularities (of time), i.e. there are different abstract views of the dynamics of the system
- it is useful to view certain actions (events) as atomic and "instantaneous" in one time band, whilst allowing them to have internal state and behaviour that takes time at a more detailed level of description

- the order (but not necessarily the time) in which events occur is important; precedence can give rise to causality
- the durations of certain actions are important, but the measuring of time must not be made artificially precise and must allow for tolerance (non-determinacy) in the temporal domain
- at each level of temporal behaviour similar phenomena are observed (e.g. cyclic/repetitive actions, deadline-driven actions, synchronous and asynchronous event handling, agreement, coordination).

The central notion in the framework is that of a time band that is defined by a *granularity*, G, (e.g. 1 minute) and a *precision*, ρ, (e.g. 3 seconds). Granularity defines the unit of time of the band; precision bounds the maximum duration of an event that is deemed to be instantaneous in its band.

A system description is not limited to a single time axis but can be given more clearly in terms of a finite set of partially ordered *bands*. System activities are placed in some band B if they engage in significant events at the time scale represented by B, i.e. they have dynamics that give rise to changes that are observable or meaningful in band B's granularity.

A complete system specification must address all dynamic behaviours. At the lowest level, circuits (e.g. gates) have propagation delays measured in nanoseconds or even picoseconds, at intermediate levels tasks/threads have rates and deadlines that are usually expresses in tens of milliseconds, at the higher levels missions can change every hour and maintenance may need to be undertaken every month. Understanding the behaviour of circuits allows the worst-case execution time of tasks to be predicted, this allows deadlines to be checked and the schedulability of whole missions to be verified. The need to argue about temporal behaviours at many levels of granularity should not by addressed by mapping all temporal properties and constraints to the finest level; the resulting model would be unmanageable. Rather, what the Timeband framework provides is the ability to use the different time granularities within the system to structure its specification into a number of distinct bands. This results in system verification being simplified to proof obligations within each band, and consistency checking between bands.

The basic means of presenting a specification over a number of bands is by relating (instantaneous) *events* in one band to *activities*, with duration, in finer bands. We use the convention that events are represented by lower case letters, and activities by upper case. Events in band B may *map* to activities that have duration at some lower (or finer) band C. An illustration of a three band system with the mapping of events to activities is shown in Figure 2, where B, C and D are timebands, x, y1, y2 and y3 are events, and Y and Z1 are activities. The curved diagram allows events at coarser bands to be represented by more detailed activities at the finer bands and thus illustrates that the finer band is a magnification of its coarser neighbour – the notion that there is more 'time' in finer bands (e.g. more milliseconds than seconds in an hour). An activity is defined to be a

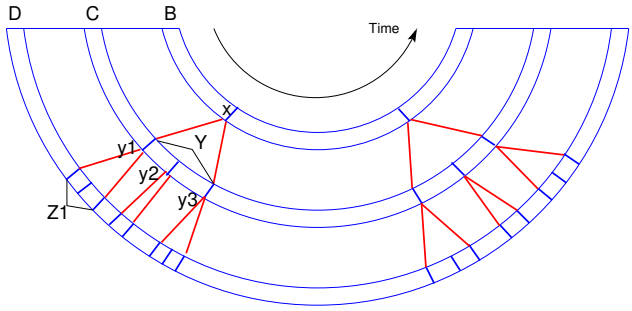

FIGURE 2. Timebands B, C and D with event x in band B implemented by activity Y in band C, which uses events y1, y2 and y3 in band C, where event y1 is implemented by activity Z1 in band D

sequence of (partially-ordered) events. The start and end of an activity A are themselves represented as events, $A\uparrow$ and $A\downarrow$, respectively.

The timeband framework [5] introduces and defines a number of notions such as simultaneous, ordered, delays, deadlines and duration. These are introduced as required in the case study. The functional and the temporal behaviours of a system are related by two concepts: accuracy and may/must.

The *accuracy* of a reading refers to an external continuously changing physical entity such as temperature.[8] With error-free behaviour a sensed value, S, is obtained by a *read* event (in some time band with precision ρ). But the exact time when this read occurs is not known precisely. The reading is said to be *accurate* to the extent $S \pm a$, where a is the maximum change that can occur to the physical phenomena in time ρ.

Within a time band all entities share the same precision and hence imprecision. This allows events to be combined (for example, two simultaneous events being combined into one). There is a common shared view of the precision of time itself within the band. Any event that is defined to occur at a particular time will have the same bound before it is declared to be too late (or too early). A single notion of precision facilitates a simple rule for composition. Different dynamic phenomena will, however, have different assessments of their accuracy. Informally, the reading of a slowly changing phenomena will be more accurate than the reading of a rapidly changing phenomena.

The accuracy issue often makes it impossible to specify a precise behaviour for a system. A specification must, however, bound the allowable behaviours despite the uncertainty. The may/must notion provides a convenient way of specifying bounds. Because a program cannot precisely determine the value and/or timing of a sampling of a value in the physical world, a specification can allow

[8]The example studied in the current paper avoids most issues of continuous behaviour by focussing on one particular subsystem. In a thermostat application, the assumed maximum rate of variation of temperature would be used to determine the required sampling frequency to stay within the required room temperature. An even more interesting example would be controlling the speed of a vehicle.

a range of possible behaviour when sensed value changes *may* be observed but oblige behaviours when sensed value changes *must* be observed. One application of this idea is given in Section 4 but the notion is general and can be deployed in many contexts.

2.4. Integrating Timebands and HJJ

The main focus and contribution of this paper is to integrate the modelling of real-time properties as represented by the timeband framework with the HJJ means of deriving a specification of the (software) control system from the overall requirements of the ES/CPS.

Each timeband has an associated set of observable variables, with the possibility that (adjacent) timebands can share some variables. In our earlier timebands work [5], the state of an observable variable associated with an event was a set of values corresponding to the values of the variable during the execution of its implementing activity. A simplification (or abstraction if you prefer) in the current paper is that an event is associated with just two states – before and after the event – and all intermediate states are hidden (ignored). This better represents the intent behind timebands that events are instantaneous within their band, and allows one to give an abstract specification of an event e in terms of its pre and post conditions, $pre\text{-}e$ and $post\text{-}e$, respectively.

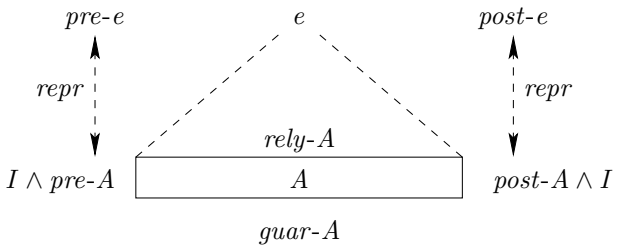

FIGURE 3. Combining rely-guarantee with timebands

If events in a timeband B are implemented by activities in a timeband C, the state space of C may contain,

- state variables of B that are shared,
- state variables of B that are shared but with their type extended to allow additional values that the variables may take on only in intermediate states (i.e. the additional values are not visible in timeband B), and
- new state variables introduced in timeband C.

The initial and final states of an event in band B correspond to the initial and final states of its implementing activity in band C via the relation $repr$ (see Fig. 3).

Each activity, A, also has pre and post conditions, $pre\text{-}A$ and $post\text{-}A$. All activities within the band share an invariant, I, that is maintained by every activity. The activity refines the event using the standard data refinement relation but augmented with a requirement that the time taken for the activity is bounded by the precision of band B (as measured in terms of the granularity of band C) – see Fig. 4.

As activities take time, they are also augmented with a rely condition, $rely\text{-}A$, stating any assumptions about the behaviour of the environment during the activity, and a guarantee condition, $guar\text{-}A$, that is a commitment of the activity over its duration. All conditions may refer to (only) state defined in the relevant time band. During activity A, as long as $rely\text{-}A$ remains true, $guar\text{-}A$ is required to be true. Other aspects of the integration are introduced by means of the case study.

3. THE SLUICE GATE EXAMPLE

In order to illustrate the notions and methods described in this paper, the 'sluice gate' case study [2] is employed. The need to irrigate an area of land could be met by the use of a number of sluice gates that control the flow of water through irrigation ditches. Any specific sluice gate will need a controller and the specification of this controller forms the focus of the case study. However the wider picture of a CPS that monitors rain fall, flood prediction, seasonal variation and even climate change gives the context for the application — this gives rise to behavioural description over a number of time bands:

Climatic To allow long term trends to be expressed.

Seasonal To allow the controls to reflect growing seasons.

Month To allow anticipated rain-fall to be expressed.

Day To allow coordination between multiple (possible failing) gates to be expressed.

Hour To allow the actual control algorithm to be expressed for a single gate.

Minute To allow the gate's main movements to be modelled.

Seconds To allow the gate's settling time to be modelled.

Millisecond To allow the behaviour of the sensors and switches to be expressed.

Microsecond To allow computation properties, such as worst-case execution time to be expressed.

It is important to note that different physical phenomena are linked to these time bands: assumptions about rainfall would have to be made at the month band but play no part once concern passes to finer levels; similarly, water is actually of no concern beyond the day band — finer bands are only concerned with movement of the gate.

In this study we focus only on the requirements for the controller of a single gate. The physical structure of such a sluice gate is represented in Figure 5. This gate is controlled by a single motor and there are sensors that indicate when the gate is at the top and when it is at the bottom. The motor is controlled by a direction indicator (up or down) and an on/off 'switch'. The expected behaviour of a single gate involves periods of time when the gate is (fully) open and periods when it is closed (no water flows). The physical

$$pre\text{-}e(s) \land repr(s, c) \land I(c) \quad \Rightarrow \quad pre\text{-}A(c) \tag{1}$$

$$pre\text{-}e(s) \land repr(s, c) \land I(c) \land post\text{-}A(c, c') \quad \Rightarrow \quad (\exists s' \cdot repr(s', c') \land post\text{-}e(s, s')) \land I(c') \land t'\text{-}t \leq \rho \tag{2}$$

FIGURE 4. Data refinement between timebands: s represents the initial state of the coarser (more abstract) band and c the state of the finer band, and s' and c' the respective final states.

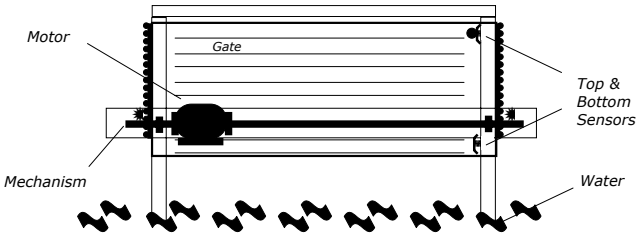

FIGURE 5. A single sluice gate

components of the ES are referred to as the MGS (motor, gate and sensors) – see top level illustrative diagram in Figure 1.

The design decision to use a motor with top and bottom sensors and relevant actuators to interface to the gate is most clearly described over three time bands:[9] Each band is defined by its granularity (G) and precision (ρ), and into each band will be placed relevant State, Events and Activities.

Hour G_h = 1 hour, ρ_h = 3 minutes.

Minute G_m = 1 minute, ρ_m = 3 seconds.

Seconds G_s = 1 second, ρ_s = 10 milliseconds.

These are the three main time bands used to organise the description of the case study. The granularity of a band does not have to correspond to a unit of measurement of time but using standard units of time allows for an easier presentation of the example.

3.1. Hour Band

The coarsest band for the single gate problem is:

Hour G_h = 1 hour, ρ_h = 3 minutes.

Remember that the $gate$ position is a physical world phenomenon: the first step of our process is to ground the system description in terms of physical world phenomena at a level of time discourse appropriate to the customer. The client could express his/her requirement in terms of water flow (per minute, per hour or per day), but having chosen a means of controlling the flow (i.e. a sluice gate) it is more natural for the requirement to be expressed in terms of the gate alternating between being open and closed for given time periods.

Once placed in the Hour band this requirement becomes unambiguous. By placing the requirement in this band it implies that issues within the precision of the band (i.e. 3 minutes in any one hour) are immaterial to the client. Note that this important and reasonable notion of approximation comes for free from the time band view; it is not necessary to write a specification in terms of minutes (or seconds!) to specify the agreed flexibility.

To give a top-level description of the control program (in the Hour band) requires one state variable and two events. These are declared together with their necessary pre and post conditions (expressed using the state variable):

$SluiceGate_h : (gate : \{\text{OPEN}, \text{CLOSED}\})$

The state of the sluice gate in the hour band, $SluiceGate_h$, has a single local component $gate$, which can be either open or closed.[10] In the pre/post conditions below $gate$ refers to the initial value of this component and $gate'$ to its final value.

event *open-gate*	**event** *close-gate*
var out $gate$	**var out** $gate$
pre $gate = \text{CLOSED}$	**pre** $gate = \text{OPEN}$
post $gate' = \text{OPEN}$	**post** $gate' = \text{CLOSED}$

Both *open-gate* and *close-gate* are events of the CPS. They are viewed, within the context of the hour band, as occurring instantaneously. Their behaviour is specified in terms of the desired behaviour of the physical gate, even though the yet-to-be-designed control software cannot directly move the gate (it must communicate with the physical components using the sensors and actuators that are introduced at a finer time band). Note that as both events have write access to $gate$ they cannot occur simultaneously (i.e. their defining activities cannot overlap). The assumption is made that in the absence of events that explicitly change state then the state does indeed not change. So between *open-gate* and *closed-gate* the state of the gate is constant.

From the definition of these events and the system state, the required schedule for the control program/activity can be derived. The schedule activity is parameterised with the number of hours the gate should be open, OG, alternating with the number of hours it should be closed, CG. The

[9] To capture details at the electronic level would require much finer bands and to specify seasonal behaviour a coarser band would be required. However, for a demonstration of the framework, we concentrate on just three adjacent bands.

[10] If more than one sluice were required, the structure containing the component gate would need to be promoted to a type.

values of these parameters are assumed to be determined by a yet coarser timeband; a typical activation might be $Schedule(2hr, 10hr)$.

activity $Schedule_1(OG, CG{:}\, hour)$
var out $gate$
pre $gate = \text{CLOSED}$
rely true
guar $(open\text{-}gate\,\mathbf{to\_next}\,close\text{-}gate) \;\Rightarrow\; dur = OG$
$(close\text{-}gate\,\mathbf{to\_next}\,open\text{-}gate) \;\Rightarrow\; dur = CG$
post $gate' = \text{CLOSED}$

In this, and following activities, there is an implied 'and' between the lines of the guarantee (and rely) conditions. Line breaks are, however, deemed to effect the priority of logical operators in the sense that each line is assumed to be in parentheses. Relies and guarantees are required to hold for all subintervals of the time interval over which the activity $Schedule_1$ operates. The interval predicate $(e_1\ \mathbf{to\_next}\ e_2)$ succeeds for an interval I, if I starts with event e_1 and ends in event e_2 and there are no intervening occurrences of e_2. The interval expression dur gives the length of the interval. The precondition refers to system start up, when *Schedule* begins. The post condition requires a $Schedule$ activity terminate only if the gate is closed; this is consistent with completing some number of complete cycles of the behaviour.

There is no informative rely condition in this time band; but the guarantee condition gives the appropriate level of separation between the opening and closing events. This specification can be refined to the following (where we only give the modified behaviour – the rely and guarantee conditions etc. are unchanged):

activity $Schedule_2(OG, CG{:}\, hour)$
var out $gate$
do $(open\text{-}gate; wait(OG); close\text{-}gate; wait(CG))^*$

Here, the behaviour of the control activity is specified after the keyword **do**. The $*$ indicates that the behaviour allows any number of complete cycles. The $wait$ primitive delays the behaviour of schedule for the designated number of hours (plus or minus 3 minutes).

The realisation of these two events by activities of a finer time band requires details of the behaviour of the hardware implementing the sluice gate mechanism: motor, gate and sensors (MGS); these are more sensibly described at a finer time band.

Before moving to a finer band, a simple extension demonstrates why it is necessary to express behaviours over a range of time granularities. It is reasonable to assume that the irrigation policy represented by the above schedule would not remain constant all year; rather it is likely to be seasonal; for example, in the growing season (summer), the gate may need to be open for longer periods of time, e.g. $Schedule(3hr, 9hr)$; in winter, there might be a minimum movement of the gate (to keep it operational); so, $Schedule(1hr, 23hr)$. Switching between the schedules would be defined in the Seasonal band with a simple precondition used to determine the appropriate active schedule.

By including bands that are more associated with human activities than computational ones it is possible to represent important aspects of the complete control program. For example, in the Seasonal band it will be necessary to determine the current season, but should this be defined by the Meteorological Office or by the consensus view of the users of the irrigation system? These control-related decisions will need to be expressed in the Seasonal band, along with any necessary assumptions (rely conditions).

3.2. Minute Band

It is typical that attention moves to a finer time band (than that of the mechanical phenomena) when the control process is to be addressed.

Minute G_m = 1 minute, ρ_m = 3 seconds.

The objective in the finer Minute band is to map (implement) the $open\text{-}gate$ and $close\text{-}gate$ events to activities. In order to do that, the necessary physical components (the sensors and the motor that drives the gate) are considered. The properties of these components are stated in terms of the guarantees they provide and the rely conditions they can assume (see MGS_m in Figure 6). The open and close gate activities are then implemented assuming the physical components will adhere to their guarantees, provided the activities do not contravene the rely conditions of the physical components.[11]

In this timeband, as well as being open or closed, the position of the gate can also be in the state BETWEEN; this acknowledges the fact that, in this band, the amount of time to move the gate between its extreme positions is significant. The variable $gate$ is inherited from the coarser band but its type is extended with the additional value BETWEEN. Furthermore, the state in this band contains components representing Boolean sensors (top and $bottom$) and an actuator ($direction$). At this level $motor$ is introduced to represent the state of the entity within the MGS that causes the gate to move. The control over the motor (via $power$ being either on or off) is necessarily introduced at the next finer time band.

$$
\begin{array}{lll}
SluiceGate_m : gate & : & \{\text{OPEN}, \text{CLOSED}, \text{BETWEEN}\} \\
\quad\quad\ \ top & : & Boolean \\
\quad\quad\ \ bottom & : & Boolean \\
\quad\quad\ \ direction & : & \{\text{UP}, \text{DOWN}\} \\
\quad\quad\ \ motor & : & \{\text{RUNNING}, \text{STOPPED}\}
\end{array}
$$

As expected, if $SluiceGate_m.gate$ is BETWEEN the open/closed state in the hour band, $SluiceGate_h.gate$, is not defined. The new events of the minute band are $set\text{-}direction$ and $set\text{-}motor$; they are defined below. But events such as $set\text{-}motor$ cannot of themselves affect the position of the $gate$, they can only write to actuators; the control system is described in terms of the signals it sends to the actuators and reads from sensors. To emphasise this

[11] The topic of fault tolerance is addressed in Section 4.

physical MGS_m
var in $direction, motor$
  out $gate, top, bottom$
pre $motor = \text{STOPPED} \wedge gate = \text{CLOSED}$
rely $direction = \text{UP} \wedge motor = \text{RUNNING} \wedge gate = \text{OPEN} \rightsquigarrow_0 motor = \text{STOPPED}$
  $direction = \text{DOWN} \wedge motor = \text{RUNNING} \wedge gate = \text{CLOSED} \rightsquigarrow_0 motor = \text{STOPPED}$
  $\boxdot(motor = \text{RUNNING}) \Rightarrow (direction' = direction)$
guar $direction = \text{UP} \wedge motor = \text{RUNNING} \rightsquigarrow_1 gate = \text{OPEN}$
  $direction = \text{DOWN} \wedge motor = \text{RUNNING} \rightsquigarrow_1 gate = \text{CLOSED}$
  $\boxdot(motor = \text{STOPPED}) \Rightarrow gate' = gate$
  $\boxdot(top \Leftrightarrow gate = \text{OPEN})$
  $\boxdot(bottom \Leftrightarrow gate = \text{CLOSED})$

FIGURE 6. Specification of behaviour of physical components in the Minutes timeband

point, notice that the control system would behave equally well if it were connected to a software simulator rather than to a physical gate.

To justify that the combined control and MGS components satisfy the overall specification, rely conditions are required about the correspondence between the sensors and the position of the physical gate.

Before tackling the implementation of the open and close gate activities, one needs to understand the properties of the given hardware components. These are represented by the behaviour of the MGS and are recorded as the relies and guarantees of the MGS_m system (see Figure 6). These reflect properties of the hardware over any time interval (at the minute band) in which they are operating correctly and hence each rely and guarantee condition is interpreted as holding over any time interval. The precondition refers to system start up, before MGS is engaged.

The notation $p \rightsquigarrow_n q$ means that, if p holds continuously then, within n time units (of the band) q will hold and then q continues to hold as long as p does, and $p \rightsquigarrow_0 q$ means "immediate" (i.e. within the precision of the band).

The first two guarantees record that the gate will open (close) within 1 minute (a minute being the granularity of the band) if the motor is running with direction up (down). The first two relies record the assumption that, if the motor is moving up (down) and the gate reaches the open (closed) position, then the motor will be stopped immediately (i.e. within the precision of this band).

The notation $\boxdot p$ is a predicate that holds for a time interval if, for all times in the interval, p holds for the state at that time. So the third rely condition requires that the direction of travel of the gate is never changed while the motor is running ($direction$ is the value at the start of the interval and $direction'$ is the value at the end). The third guarantee clause in the definition means that, in any interval in which motor continuously has the value STOPPED, the value of the $gate$ does not change. The final two guarantee conditions state that the top (bottom) sensor being on corresponds to the $gate$ being at the open (closed) position.

Note the particular physical component identified in this specification relies on three necessary properties of the controlling software. The first two ensure that motor is stopped immediately the gate reaches the top (or bottom). By placing this rely condition in the minute band this need for 'immediate' action can be implemented by an activity (in a finer band) that has a duration of no more than ρ_m (i.e. 3 seconds). The final requirement is that the controlling software never changes the direction of travel of the gate while the motor is engaged (because that may damage the gate mechanism).

In terms of the performance of the MGS, it guarantees to lift the gate within 1 minute. This easily satisfies the implied requirement, from the hour band, to open the gate within ρ_h (i.e. 3 minutes). Obviously the decision to deploy this control system must include a check that the physical MGS component is in conformance with the rely conditions. Different MGS may have different requirements and capabilities.

As noted earlier, the new events introduced within the minute band are $set\text{-}direction$ and $set\text{-}motor$.

event $set\text{-}direction(d\colon \{\text{UP}, \text{DOWN}\})$
var out $direction$
  in $motor$
pre $motor = \text{STOPPED}$
post $direction' = d$

event $set\text{-}motor(v\colon \{\text{RUNNING}, \text{STOPPED}\})$
var out $motor$
post $motor' = v$

Note that as one of these events has write access to $motor$ and the other read access they are prevented from occurring simultaneously (and their associated activities hence cannot overlap).

The code for the activity $Open\text{-}Gate_1$ in Fig. 7 in the minute band corresponding to the $open\text{-}gate$ event is

activity $Open\text{-}Gate_1$
var in $gate, top$
  out $direction, motor$
pre true
invariant $motor = \text{STOPPED}$
rely $direction = \text{UP} \wedge motor = \text{RUNNING} \rightsquigarrow_2 gate = \text{OPEN}$
  $\boxdot(motor = \text{STOPPED}) \Rightarrow gate' = gate$
  $\boxdot(top \Leftrightarrow gate = \text{OPEN})$
guar $direction = \text{UP} \wedge motor = \text{RUNNING} \wedge gate = \text{OPEN} \rightsquigarrow_0 motor = \text{STOPPED}$
  $direction = \text{DOWN} \wedge motor = \text{RUNNING} \wedge gate = \text{CLOSED} \rightsquigarrow_0 motor = \text{STOPPED}$
  $\boxdot(motor = \text{RUNNING}) \Rightarrow direction' = direction$
post $gate' = \text{OPEN}$

FIGURE 7. Activity $Open\text{-}Gate_1$

obtained by the application of a number of HJJ/refinement steps. An overview of the relationships is outlined in Fig. 3. First the event itself is refined into an activity that can take advantage of the conditions that are guaranteed by MGS_m. The relationship between the states at the two levels, $repr$, in this case is just that the values of $gate$ correspond. Refinement allows weakening of the precondition – see (1) in Fig. 4 – from $(gate = \text{CLOSED})$ to **true**, (i.e. it can be omitted). The postcondition may also be strengthened (2), although it remains unchanged in this case. It has additional variables defined in the minute band state. Both the precondition and postcondition are implicitly strengthened with the invariant $(motor = \text{STOPPED})$. Note the invariant is assumed to hold initially via the precondition of MGS_m (Fig. 6).

The physical component MGS_m operates in parallel with the activity and hence $Open\text{-}Gate_1$ can rely on the guarantee of MGS_m; in this case two conditions in the guarantee are not needed and are omitted from the rely of $Open\text{-}Gate_1$ (although they are needed for $Close\text{-}Gate_1$ which is not detailed here). Symmetrically, the activity must guarantee the rely of MGS_m. Together these refinements result in the first definition of the activity $Open\text{-}Gate_1$ in Figure 7. The definition of $Close\text{-}Gate$ is similar.

Subsequent steps are within the minute band and are used to remove direct access to state components that cannot be accessed by the code. This is the process loosely described above as subtracting (from the requirements in the physical world) information about mechanical components in order to derive a specification of the control system. For example, in $Open\text{-}Gate_2$, $gate$ is no longer referenced but the sensor variable top is. The rely condition below is implied by the rely of $Open\text{-}Gate_1$, and the guarantee below implies the guarantee of $Open\text{-}Gate_1$, assuming the rely of $Open\text{-}Gate_1$ holds.

activity $Open\text{-}Gate_2$
var in top
  out $direction, motor$
pre true
invariant $motor = \text{STOPPED}$
rely $direction = \text{UP} \wedge motor = \text{RUNNING} \rightsquigarrow_2 top$
  $\boxdot(motor = \text{STOPPED}) \Rightarrow top' = top$
guar $\boxdot(motor = \text{RUNNING} \Rightarrow direction = \text{UP})$
  $motor = \text{RUNNING} \wedge top \rightsquigarrow_0 motor = \text{STOPPED}$
post top'

The guarantees can now be used to define the required behaviour of the activity. Note that $motor$ is still referenced at this level (via the $set\text{-}motor$ event); in the Seconds band this is replaced by use of an I/O routine that turns power on and off to the motor.

We are now able to implement the activity at the Minute band in terms of events in this band. The event $set\text{-}direction(\text{UP})$ does not break the rely condition of MGS_m, not to switch the direction while the motor is running, because the motor is assumed to be stopped initially.

activity $Open\text{-}Gate_3$
var in top
  out $direction, motor$
do $set\text{-}direction(\text{UP}) \vec{\#} set\text{-}motor(\text{RUNNING});$
  $await(top) \vec{\#} set\text{-}motor(\text{STOPPED})$

For events e and f, $e \vec{\#} f$ requires that the behaviour described by "$e; f$" happens within the precision of the band; $e \vec{\#} f$ can be thought of as a single event with a behaviour equivalent to the sequential composition of e and f. In the second use of the simultaneous operator above the first operand is the activity $await(top)$; the simultaneity requirement is then on the end event of the await, $await(top)\downarrow$, and the event $set\text{-}motor(\text{STOPPED})$.

As the $Open\text{-}Gate$ activity is mapped from an event in the hour band there is an implicit requirement that the entire behaviour will take no more than ρ_h, i.e. 3 minutes.

physical MGS_s
var in $power, direction$
  out $motor$
pre $motor = \text{STOPPED} \wedge power = \text{OFF}$
rely $\boxdot(motor \neq \text{STOPPED}) \;\Rightarrow\; (direction' = direction)$
guar $power = \text{ON} \rightsquigarrow_1 motor = \text{RUNNING}$
  $power = \text{OFF} \rightsquigarrow_1 motor = \text{STOPPED}$
  $\boxdot(power = \text{OFF}) \wedge motor = \text{STOPPED} \;\Rightarrow\; motor' = \text{STOPPED}$
  $\boxdot(motor = \text{STOPPED} \;\Rightarrow\; power = \text{OFF})$

FIGURE 8. Specification of the physical motor, gate and sensors (MGS) in the Seconds band

In this code $await(top)$ implies an interval of time that is terminated by the recognition, in the control program, that top is true. If top is true for a short interval of time (less than ρ_m) then the interval *may* be terminated, but if top is true for a duration equal or greater than ρ_m then it *must* terminate within that interval.

The rely condition expressed within $Open\text{-}Gate_2$ ensures that the maximum time between the events $set\text{-}direction(\text{UP})$ and $await(top)\downarrow$ is at most 2 minutes (plus 2 ρ_m).

3.3. Seconds Band

The overall schedule sits naturally in the hour time band; the physical movement of the gate was most appropriately described at the minute band; detailed interactions with the control component are best described at a finer time band. It is fairly typical that this will be at least as fine as seconds.

Seconds G_s = 1 second, ρ_s = 10 milliseconds.

This is the appropriate time band to note that for the chosen MGS the motor does not stop rotating instantaneously (as observed in the Minute band). Rather there is an additional short time interval in which the direction of travel should not be changed. The state $motor$ has two additional possible values, STARTING and STOPPING, representing the additional phases of motor operation. The control software cannot directly control the motor, rather it must manipulate an object in the interface (a boolean variable, $power$), local to the implementation, that controls whether power has been applied to the motor.

$$\begin{array}{lll} SluiceGate_s : motor & : & \{\text{RUNNING}, \text{STOPPED}, \\ & & \text{STARTING}, \text{STOPPING}\} \\ \quad power & : & \{\text{ON}, \text{OFF}\} \\ \quad direction & : & \{\text{UP}, \text{DOWN}\} \end{array}$$

The additional behaviour of the equipment in the seconds timeband is given by MGS_s in Figure 8. It strengthens the rely of MGS_m to ensure the direction is only changed while the motor is fully stopped (not starting, stopping or running). Power on (off) guarantees to get the motor running (stopped) within 1 second. If the motor is initially stopped and the power is continuously off, the motor remains stopped. For the motor to be fully stopped the power must be off.

It is important to understand that MGS_s is not a refinement of MGS_m: the defined behaviour of the MGS satisfies both MGS_m and MGS_s specifications.

The two activities can now be defined, firstly $Set\text{-}Direction$:

activity $Set\text{-}Direction_1(d\colon \{\text{UP}, \text{DOWN}\})$
var out $direction$
  in $power, motor$
pre $motor = \text{STOPPED} \wedge power = \text{OFF}$
rely $\boxdot(power = \text{OFF}) \wedge motor = \text{STOPPED} \;\Rightarrow\; motor' = \text{STOPPED}$
guar $\boxdot(motor \neq \text{STOPPED}) \;\Rightarrow\; (direction' = direction)$
post $direction' = d$

The precondition is strengthened with $power = \text{OFF}$ based on the last guarantee of MGS_s. This is satisfied by simple code consisting of a single assignment.

activity $Set\text{-}Direction_2(d\colon \{\text{UP}, \text{DOWN}\})$
var out $direction$
do $direction \leftarrow d$

The specification for $Set\text{-}Motor$ is similarly derived:

activity $Set\text{-}Motor_1(v\colon \{\text{RUNNING}, \text{STOPPED}\})$
var out $power, motor$
rely $power = \text{ON} \rightsquigarrow_1 motor = \text{RUNNING}$
  $power = \text{OFF} \rightsquigarrow_1 motor = \text{STOPPED}$
guar $\boxdot(motor \neq \text{STOPPED}) \;\Rightarrow\; (direction' = direction)$
post $motor' = v$

Once more, it is easy to write code that satisfies the specification.

activity $Set\text{-}Motor_2(v\colon \{\text{RUNNING}, \text{STOPPED}\})$
var out $power$
do $power \leftarrow$ (**if** $v = \text{RUNNING}$ **then** ON **else** OFF);
  $wait(2)$

The $wait(2)$ is required to ensure that when this activity is completed the motor is RUNNING/STOPPED (not just STARTING/STOPPING); the event *set-motor* has this as a post-condition. The wait of 2 seconds is conservative because the motor is assumed to stop in 1 second.

3.4. Summary

Although the description above has used some notation that has not been fully formalised, all steps can be represented as proof obligations about which one can reason. The overall result is the following defined behaviours (and their associated rely conditions) for the control software. Note that the complete specification incorporates and requires all three bands, it has not been refined down to just the Seconds band.

In the Hour band:

activity $Schedule(OG, CG{:}\, hour)$
var out $gate$
pre $gate = \text{CLOSED}$
do $(open\text{-}gate; wait(OG); close\text{-}gate; wait(CG))^*$

In the Minute band:

activity $Open\text{-}Gate$
var in top
  out $direction, motor, gate$
invariant $motor = \text{STOPPED}$
rely $direction = \text{UP} \wedge motor = \text{RUNNING} \leadsto_2 gate = \text{OPEN}$
  $\boxdot(motor = \text{STOPPED}) \;\Rightarrow\; gate' = gate$
  $\boxdot(top \;\Leftrightarrow\; gate = \text{OPEN})$
do $set\text{-}direction(\text{UP}) \overset{\rightarrow}{\#} set\text{-}motor(\text{RUNNING});$
  $await(top) \overset{\rightarrow}{\#} set\text{-}motor(\text{STOPPED})$

activity $Close\text{-}Gate$
var in $bottom$
  out $direction, motor, gate$
invariant $motor = \text{STOPPED}$
rely $direction = \text{DOWN} \wedge motor = \text{RUNNING} \leadsto_2 gate = \text{CLOSED}$
  $\boxdot(motor = \text{STOPPED}) \;\Rightarrow\; gate' = gate$
  $\boxdot(bottom \;\Leftrightarrow\; gate = \text{CLOSED})$
do $set\text{-}direction(\text{DOWN}) \overset{\rightarrow}{\#} set\text{-}motor(\text{RUNNING});$
  $await(bottom) \overset{\rightarrow}{\#} set\text{-}motor(\text{STOPPED})$

And in the Seconds Band:

activity $Set\text{-}Direction(d{:}\,\{\text{UP}, \text{DOWN}\})$
var in $motor, power$
  out $direction$
pre $motor = \text{STOPPED} \wedge power = \text{OFF}$
rely $\boxdot(power = \text{OFF}) \wedge motor = \text{STOPPED} \;\Rightarrow\; motor' = \text{STOPPED}$
do $direction \leftarrow d$

activity $Set\text{-}Motor(v{:}\,\{\text{RUNNING}, \text{STOPPED}\})$
var out $motor, power$
rely $power = \text{ON} \leadsto_1 motor = \text{RUNNING}$
  $power = \text{OFF} \leadsto_1 motor = \text{STOPPED}$
do $power \leftarrow$ (**if** $v = \text{RUNNING}$ **then** ON **else** OFF);
  $wait(2)$

This 3-band specification has a number of implicit timing constraints: the *Open-Gate* and *Close-Gate* activities must always complete in 3 minutes, and the *Set-Direction* and *Set-Motor* activities must always complete within 3 seconds. In addition, if the *top* (or *bottom*) sensor comes on for more than the band precision of 3 seconds then it must be recognised, and the *Set-Motor* activity also completed within that 3 seconds.

To complete this specification the abstract events of assigning values to *direction* and *power* and the internal events of $await(top)$ and $await(bottom)$ would need to be mapped to activities (code) that is represented at a microsecond band. This would include register manipulation and perhaps interrupt handling. However for the purposes of this paper this partial specification is sufficient.

The completed specification would guarantee that the requirements of the customer, and the constraints of the physical components of the ES/CPS are satisfied if all the rely conditions from the environment and physical components remain true. Resilience is, however, more than just satisfying functional and temporal requirements. A key aspect of resilience is fault tolerance which we address in the following section. An important issue here is what happens if one or more rely conditions fail.

4. FAULT TOLERANCE

Fault tolerance is a crucial aspect of any resilient ES or CPS. Physical components are inevitably subject to failure;[12] a link to the standard defence of hardware redundancy is made in Section 4.4. A control system cannot achieve its desired behaviour in the presence of arbitrary failures of components but a resilient system should not behave in a completely uncontrolled way when (minor) failures occur.

It is important to remember that the overall specification of a control system is the combination of all of the time bands: time bands are not to be viewed as refinements. Failures can however show themselves at any time band and, in many cases, have the interesting property that a failure at one band has to be handled as a fault at the next coarser band.

In the framework proposed in this paper, the assumptions about the behaviour of the physical components of a system have been recorded as rely conditions. The approach to increased fault tolerance is to challenge each rely condition and where appropriate (and possible) record nested rely/guarantee conditions that –as well as specifying ideal behaviour in the absence of component failures– also record weaker promises when components fail to respect the ideal rely conditions.

[12] The terms fault, error and failure from [25] are used here.

The approach proposed here is to question rely conditions systematically and see what can be achieved even if they do not hold. Given the importance of time bands in this paper, attention is also given to where an activity might take longer than the precision of the band of the corresponding event.

The position taken to monitoring of rely conditions is, however, that extra reporting of failures has to be specified. It is, in general, not expected that a control program will monitor for possible deviation from its rely conditions. This situation can be compared with Michael Jackson's telling rejection[13] of checking pre conditions: if a program is to test its pre condition, good decomposition suggests decomposing to a checking routine and one that executes the main function; the latter has an identical pre condition to that of the whole; which could result in an infinite regress. Assumptions –be they pre or rely conditions– are invitations to the developer to ignore certain possible situations; it is the obligation of the deployer to ensure (i.e. prove) that the context will meet the assumptions. Of course, there is no objection to specifying diagnostic messages but such extra checks must be part of the specification.

In the sub-sections below we first consider explicit rely conditions that are part of the specification; then the implicit temporal rely conditions that results from the use of events in the timeband framework. We then consider the use of the may/must notion within health monitoring code. Finally we give a brief indication of how hardware redundancy schemes fit into the picture.

4.1. Challenging the rely conditions

In this section we revisit the specification derived in Sect. 3 and address each rely condition in each time band. We note that, in general, a single fault may lead to multiple broken rely conditions and that a single broken rely condition could have multiple possible faults that can cause it. For a safety-critical system the necessary safety-case often contains arguments (about the system's integrity) based on FTA (Fault Tree Analysis) and FMEA (Failure Modes and Effects Analysis) [26]. Rely conditions provide a focus for FMEA (and to some extent FTA) by being explicit about the assumptions made for correct behaviour.

Section 3.4 records every rely condition. We now consider each in turn.

In the Hour band the rely is unbreakable (being just **true** it can never be **false**). No further action is required.

In the Minute band (and concentrating on just the *Open-Gate* activity as similar augments apply to *Close-Gate*) we have for $rely\text{-}Open\text{-}Gate_1$:

$$direction = \text{UP} \wedge motor = \text{RUNNING} \rightsquigarrow_2 gate = \text{OPEN}$$
$$\boxdot(motor = \text{STOPPED}) \;\Rightarrow\; gate' = gate$$
$$\boxdot(top \;\Leftrightarrow\; gate = \text{OPEN})$$

If the first condition does not hold, the motor fails to open the gate in the time expected. This could be due to the motor being broken, power not being supplied or the gate being jammed. It is impossible for the control software to distinguish between these causes. Although the specification of the MGS in the Minute band guarantees to open the gate in one minute, the rely condition for the *Open-Gate* activity allows two minutes. An even weaker rely could therefore be defined:

$$direction = \text{UP} \wedge motor = \text{RUNNING} \rightsquigarrow_3 gate = \text{OPEN}$$

With this weaker rely condition there is now a two minute gap between what the MGS guarantees and the software requires. This could allow extra functionality to be added to the software, for example to attempt to remove possible jamming by moving the gate down and up again (this would perhaps be more useful within *Close-Gate* where jamming is more likely). Any extra functionality would of course require the specification to be modified. If the specification is not so extended then after 3 minutes (the precision of the Hour band in which the *open-gate* event is defined) then there will be an *open-gate-fail* event in the Hour band (see discussion below in Section 4.2).

The second clause of $rely\text{-}Open\text{-}Gate_1$ embodies the assumption that only the motor can move the gate. In particular that an open gate will not (under the force of gravity) slowly sink to the bottom and thereby reduce the amount of irrigation water passing the sluice gate. A response to this potential failure could be to check periodically (in the Seconds band) that *top* remains true for the entire time that the gate is open and the motor is stopped. Again, this requirement whose role is to add to the resilience of the CPS necessarily leads to an extended specification.

The final clause concerns the sensor. If this is malfunctioning (which the control software cannot know) then the controller cannot distinguish between motor failure and sensor failure. If it is the sensor then the gate may be open (at the top) but the control software has not turned off the motor. This could lead to the motor burning out. To prevent this eventuality two strategies can be adopted. First the reliability of the sensor can be improved — perhaps by replication; the use of hardware redundancy and replication is covered in Section 4.4. Assuming that voting between sensors is handled separately, this approach has no impact on the specification of the software (including the rely conditions). The second strategy is to introduce nested rely/guarantee conditions. For example, the specification of the software (in the Minute band) could be extended to say that the motor never runs for more than 3 minutes in any 30 minutes. We now have a weaker rely condition (**true**) but a guarantee that ensures that the motor does not burn out. Of course with the defined system that only has sensors at the top and bottom then the actual position of the gate when the power is cut will be unknown. This needs to be reflected in the Hour band when the *open-gate-fail* event occurs (see discussion in Sect. 4.2).

This completes consideration of the Minute band. For the Seconds band we have the following rely conditions; in *Set-Direction*:

$$\boxdot(power = \text{OFF}) \wedge motor = \text{STOPPED} \;\Rightarrow\; motor' = \text{STOPPED}$$

[13] Private verbal communication.

and in $Set\text{-}Power$:

$$power = \text{ON} \rightsquigarrow_1 motor = \text{RUNNING}$$
$$power = \text{OFF} \rightsquigarrow_1 motor = \text{STOPPED}$$

The first clause only really requires the power supply to the motor to be appropriately wired. In the absence of any security issues then a high level of confidence can be assigned to this assumption. Nevertheless it is important that the rely condition has been explicitly stated.

The second set of clauses can again suffer from complete functional failure or timing overrun. To check that the motor is running (without the introduction of further sensors) could be accomplished by monitoring the arrival time at the top (or bottom). Again however, the failure of the gate to open (or close) could be due to a number of reasons that the control software cannot distinguish between. The failure to stop the motor would need to be investigated at a level below the Seconds band where the code for power management is to be found – but this is beyond the scope of this study.

Note the clauses in $rely\text{-}MGS_m$ and $rely\text{-}MGS_s$ play a different role: the control system is required to make sure that these are not violated. Thus they become part of the guarantee conditions for the control system.

A full treatment of the likelihood of different failure modes requires use of a stochastic model which is beyond the aims of the current paper but adding probabilities is compatible with our framework.

4.2. Timing failures

As noted in the above discussion it is possible for an event to fail. In particular any event (or collection of synchronous events) that fails to occur within the precision of the band in which they are defined with result in a 'fail' event. Indeed for any event e there is an implicit event $e\text{-}fail$, with the post condition of $e\text{-}fail$ being weakened to **true**. At run-time the environment will determine if e or $e\text{-}fail$ occur. A system is built on the assumption that each event will terminate without failure; only in the consideration of fault tolerance will the possibility of $e\text{-}fail$ be investigated.

So event $open\text{-}gate$ in the Hour band has an associated event $open\text{-}gate\text{-}fail$. This event occurs when either the $Open\text{-}Gate$ activity in the Minute band fails to terminate within 3 minutes (the precision of the Hour band), or the activity itself signals failure. The occurrence of this failure event could be specified to be actually generated by the software running within the minute band; it could then cause the control schedule to be abandoned, and perhaps a warning signal/light/horn to be activated (in the Hour band).

A further consequence of a timing failure is when a finer-band state, that does not correspond to any coarser-band state, is occupied for longer than the precision of the band within which the finer state is defined. So, for example, in the Hour band the gate is either OPEN or CLOSED; but in the Minute band it can also be in the state BETWEEN. The timeband model requires that the longest time this state can be occupied is 3 minutes (the precision of the Hour band) per hour. But a failure (e.g. motor failure) can leave the gate in the BETWEEN state indefinitely.

To cater for this failure behaviour the type of such state variables in the coarser band is expanded with an *undefined* value, $\perp$, which is used to represent that the state is undefined – this corresponds to it being any value in the lower band (not just BETWEEN in the $open\text{-}gate$ example).

To give a further example from the case study: in the Minute band there are only two possible states for the motor, RUNNING and STOPPED, but in the finer Seconds band the phases of STARTING and STOPPING the motor are also visible (see Figure 9). The variable $SluiceGate_s.motor$ being either STARTING or STOPPING for longer than the precision ρ_m corresponds to $SluiceGate_m.motor = \perp$ in the coarser band. The weaker post condition **true** holds for any value of $SluiceGate_m.motor$, including $\perp$. Because the interpretation of $\perp$ is that the value is undefined, it rarely makes sense to explicitly use $\perp$ in post conditions.

4.3. Applying the may/must notion to monitoring

There are cases where indirect inferences can be used to indicate erroneous behaviour of the physical components. This can then be used to monitor the run-time health of the system. One such case is used here to illustrate both an interesting timeband example and the may/must approach which has quite general applicability.

It is an indirect consequence of a physical world impossibility that the top and $bottom$ sensors should never both be on at the same time.

$$\begin{aligned} top \wedge bottom \quad &\Leftrightarrow \quad gate = \text{OPEN} \wedge gate = \text{CLOSED} \\ &\Leftrightarrow \quad false \end{aligned}$$

The notion of "the same time" is however interesting. One point is that a control program cannot read both sensors at exactly the same point in time and hence it is impossible to test this assertion. However, given that the sensors should reflect the gate position that cannot change quickly, it would be suspicious if both sensors were on even within a relatively short period of time. What could be happening is some sort of sensor flicker caused by a bad connection and such phenomena might well be discussed at the millisecond time band.

A control program should be allowed to raise an alert if top and $bottom$ are sensed in a short period of time and it ought raise an alarm if both are sensed over a long period of time. By placing the health monitor in some band with precision ρ, the implication is that if top and $bottom$ are both on in an interval of duration greater than ρ it must identify the fault; for an interval less than ρ it may.

The approach of may/must specification is useful in many situations where it is necessary to allow non-determinism but also to bound the degree of flexibility. As here, it is particularly useful with time bands.

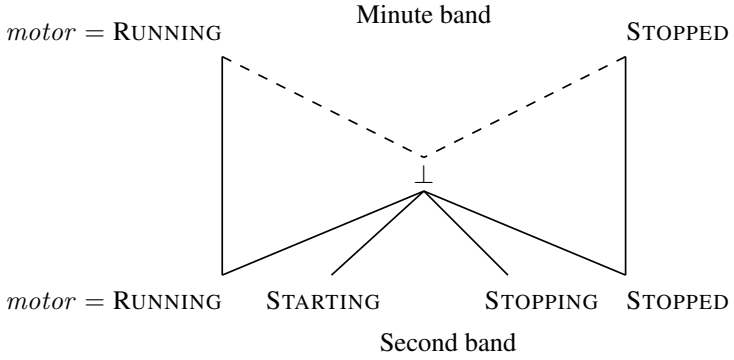

FIGURE 9. Mapping states between bands and handling undefined values

4.4. Using Hardware Redundancy

This section does not aim to offer new material; rather it relates the proposed approach to established ideas. Improving the resilience of the system can be handled by incorporating more reliable sensors or multiple sensors, etc.

The final rely clause of $Open\text{-}Gate_1$ is:

$$\boxdot(top \;\Leftrightarrow\; gate = \text{OPEN})$$

This records an assumption about the relationship between real world phenomena and the state of sensors. Such assumptions are essential in reasoning that the combined control/MGS systems will satisfy the overall requirement which is expressed in terms of phenomena of the physical world: insulating the control system from the physical assumptions is key to the "HJJ" idea. Notice that a devious MGS system could fail to move the gate but send timely sensor signals to the control software and the latter would continue to operate according to its specification. Clearly, in this case, the whole system would not meet its specification because its relies would not be satisfied. Also, if the meanings of direction and top/bottom were simultaneously reversed, it would appear to operate correctly but go up instead of down and vice versa.

Leaving aside such malicious behaviour, there are numerous physical failures that could result in the above rely clause not being satisfied. The most obvious one is probably that the sensor fails. The standard way to increase dependability in such situations is to employ redundancy: for a safety critical system, a designer might add an extra sensor or even triple-modular redundant sensors. Such expense is probably not appropriate for our sluice gate.

Multiple sensors failing is less likely than the gate being jammed leading to the system being unable to determine whether the gate is at the top or not. Of course, two sensors could fail together due to a common mode failure (such as both wires being severed at the same time because they follow a similar path).

Another safety precaution might be to introduce a heat sensor on the motor — this option is not pursued here. Neither is the addition of a water flow monitor; this would be a significant change to the specification and would introduce further assumptions (about the reliability and fidelity of this additional monitor).

5. RELATED WORK

Many other researchers address issues around CPS (and some earlier research referred to as "hybrid systems" is relevant); a particularly useful recent reference is [1]. This section refers only to research that is most closely related to our approach; for a wider survey see the reports of the EU-funded CyPhERS[14] action or the CPSoS project.[15]

Coping with time One major distinction of our approach from everything else mentioned in this section is our use of time bands as a way of obviating the need to specify all time dependent values (often continuously varying) at a single time granularity. The only other approach that attempts to deal with different time granularities is that of Corsetti [27, 28] (but they ultimately map everything down to the lowest level).

Focus Broy and Stølen [29] introduce an approach to real-time systems based on timed traces with specifications in the form of assumptions (relies) and guarantees similar to those used here but they do not make use of layering based on timebands or implementing events as activities.

Hybrid automata The main abstraction mechanism used in hybrid automata is the functional block that corresponds to a component with a more abstract specification that is then implemented by a network of components [30]. While the partitioning into components has similarities to the approach taken here, hybrid automata do not take advantage of partitioning the system into time bands, instead, everything is working within the one notion of time.

Duration Calculus The Duration Calculus [31] allows specification of behaviour over time intervals and its notations could be used to express some of the detailed properties given in our example. It does not directly support

[14] Cyber-Physical European Roadmap & Strategy — see http://www.cyphers.eu

[15] http://www.cpsos.eu/wp-content/uploads/2015/02/D2-4-State-of-the-art-and-future-challenges-in-cyber-physical-systems-of-2.pdf

splitting the specification over multiple time bands.

Domain engineering Dines Bjørner has championed "Domain Engineering" in a series of publications. This approach (for example [32]) certainly places a laudable emphasis on the importance of understanding the physical world with which the computer system has to interact. It does, however, prompt the modelling of that world where our approach tries to minimise this by recording only the essential assumptions about how that world behaves. To take the example of a system for plotting possible positions over time of aircraft: we would record rely conditions about the relationship of messages transmitted to the pilot and the expected rate of climb; we would avoid discussing a model of the fluid dynamics of air over wing surfaces etc. When considering the role of human agents within a system, it becomes mandatory to record (only) assumptions about their behaviour since modelling human minds and physical capabilities is clearly unachievable.

Harel's state charts David Harel's hierarchical state charts [33] would seem relevant. Within a state chart, a single state can expand to include an internal state machine with transitions between internal states. The main difference in our approach is that, whereas Harel expands a single state to a state machine, we refine the state *transitions* to introduce more detailed behaviour to implement the transition, including additional intermediate states. For example, a start chart representing the sluice gate at the hour band could be represented by two states OPEN and CLOSED transitions between them in opposite directions labeled with events $open$ and $closed$, but while one can decompose a state, e.g. OPEN, to give the internal behaviour while the gate is open, there is no way to decompose the $open$ event into a more complex activity as we do using the approach outlined in the paper. In summary, state charts focus on decomposing states, while our approach focuses on decomposing transitions.

Four variable model David Parnas and Jan Madey [34] recognised –on the one hand– the distinction between physical "monitored" values (m) and the "inputs" visible via sensors (i) and –on the other hand– "output data items" (o) and their effect on "controlled" values (c) in the physical world. They distinguish three relations $NAT(m, c)$, $IN(m, i)$ and $OUT(o, c)$, which correspond to the description of the physical world and its relationship to the inputs and outputs of the control software, whereas we group these into a single description of the physical component (e.g. MGS) but explicitly distinguish between the relies and guarantees of the physical component, and partition the specifications into timebands to avoid the complexity of a monolithic specification.

Event-B Jean-Raymond Abrial uses Event-B [11] by starting off with a abstract representation and properties of operations at that level. He then adds state and "refines" operations to specify more of their behaviour. However, his events are discrete and atomic and hence he cannot represent rely and guarantee conditions on continuous variables.

Teleo-reactive programs Earlier work [35, 36, 37] addressed describing hybrid systems using Nilsson's teleo-reactive programming approach [38, 39]. It allowed different components to be specified within different time bands, which determined, for example, how the guards of a teleo-reactive program are to be interpreted in a "sampling" logic [40].

6. FURTHER WORK AND CONCLUSIONS

The starting point for deriving the specification of a control system is taken to be a description of the desired behaviour of the complete Cyber-Physical System (CPS) in its physical environment. This paper illustrates the benefits that can be gained by specifying behaviours at a number of different time granularities.

The proposed approach is built upon the mapping of events (with pre/post conditions) in one time band to activities (with pre and post conditions plus rely and guarantee conditions) in a time band with finer granularity. A process based on refinement and HJJ is used to transpose a specification that, usefully, refers to elements of the physical world, to one that only accesses local state and elements of the software/hardware interface. The result is a complete specification of the required behaviour of the software control component of the CPS.

Much of course remains to be done. In evolving the presented material, other examples have already been considered. Even a humble thermostat example shows both the useful separation of the external objectives in terms of real world phenomena from the detail of the control program (it also illustrates the importance of recording assumptions about the potential rate of change of external factors). More substantial examples that have motivated our work include the mine pump [41], vehicle cruise control (see [6, §3]) and the iFACTS air traffic system[16]. Rather than work on these *post facto*, we would prefer to get involved with a new application.

Related to this point, is that it is worth repeating that the proposed combination of ideas is to be viewed as a framework. Different industrial contexts will be committed to using their own standard notations and it is not our objective to offer a single notation that must be adopted to use our framework.

This, of course, connects to the question of tool support. There are efforts in both Queensland and Newcastle to provide support in Isabelle for rely/guarantee reasoning [42, 43].

One of the aims of the approach is to increase the resilience of CPS by allowing the assumptions on which the control system is built to be challenged and, where appropriate, weaker guarantees to be developed from weaker assumptions (rely conditions). Future work will extend

[16] See https://www.bcs.org/upload/pdf/formal-methods-100113.pdf

this aspect of the framework by formalising how a CPS executing in a degraded mode can return to full functionality.

ACKNOWLEDGEMENTS

The authors are grateful for useful comments from the anonymous referees of an earlier draft of this material.

This research was supported by the UK EPSRC "Strata" Platform Grant EP/N023641/1 and Australian Research Council (ARC) Discovery Grant DP190102142.

REFERENCES

[1] Platzer, A. (2018) *Logical Foundations of Cyber-Physical Systems*. Springer-Verlag.

[2] Jackson, M. (2000) *Problem Frames: Analyzing and structuring software development problems*. Addison-Wesley.

[3] Hayes, I., Jackson, M., and Jones, C. (2003) Determining the specification of a control system from that of its environment. In Araki, K., Gnesi, S., and Mandrioli, D. (eds.), *FME 2003: Formal Methods*, LNCS, **2805**, pp. 154–169. Springer Verlag.

[4] Jones, C. B., Hayes, I. J., and Jackson, M. A. (2007) Deriving specifications for systems that are connected to the physical world. In Jones, C. B., Liu, Z., and Woodcock, J. (eds.), *Formal Methods and Hybrid Real-Time Systems*, LNCS, **4700**, pp. 364–390. Springer Verlag.

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

[6] Romanovsky, A. and Thomas, M. (2013) *Industrial deployment of system engineering methods*. Springer.

[7] Davis, R., Bate, I., Bernat, G., Broster, I., Burns, A., Colin, A., Hutchesson, S., and Tracey, N. (2018) Transferring Real-Time Systems Research into Industrial Practice: Four Impact Case Studies. In Altmeyer, S. (ed.), *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*, Dagstuhl, Germany, Leibniz International Proceedings in Informatics (LIPIcs), **106**, pp. 7:1–7:24. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[8] Jones, C. B. (1990) *Systematic Software Development using VDM*, second edition. Prentice Hall International.

[9] Hayes, I. (ed.) (1993) *Specification Case Studies*, second edition. Prentice Hall International, Englewood Cliffs, N.J., USA.

[10] Abrial, J.-R. (1996) *The B-Book: Assigning programs to meanings*. Cambridge University Press.

[11] Abrial, J.-R. (2010) *The Event-B Book*. Cambridge University Press, Cambridge, UK.

[12] Jones, C. B. (1981) Development Methods for Computer Programs including a Notion of Interference. PhD thesis Oxford University. Printed as: Programming Research Group, Technical Monograph 25.

[13] Jones, C. B. (1983) Specification and design of (parallel) programs. *Proceedings of IFIP'83*, pp. 321–332. North-Holland.

[14] Jones, C. B. (1996) Accommodating interference in the formal design of concurrent object-based programs. *Formal Methods in System Design*, **8**, 105–122.

[15] Hayes, I. J., Jones, C. B., and Colvin, R. J. (2014) Laws and semantics for rely-guarantee refinement. Technical Report CS-TR-1425. Newcastle University.

[16] Jones, C. B., Hayes, I. J., and Colvin, R. J. (2015) Balancing expressiveness in formal approaches to concurrency. *Formal Aspects of Computing*, **27**, 465–497.

[17] Hayes, I. J. and Jones, C. B. (2018) A guide to rely/guarantee thinking. In Bowen, J., Liu, Z., and Zhan, Z. (eds.), *Engineering Trustworthy Software Systems – Second International School, SETSS 2017* LNCS. Springer-Verlag.

[18] Morgan, C. C. (1994) *Programming from Specifications*, second edition. Prentice Hall.

[19] Back, R.-J. R. and von Wright, J. (1998) *Refinement Calculus: A Systematic Introduction*. Springer, New York.

[20] Hoare, C. A. R. (1969) An axiomatic basis for computer programming. *Communications of the ACM*, **12**, 576–580, 583.

[21] Wei, K., Woodcock, J., and Burns, A. (2012) Modelling temporal behaviour in complex systems with timebands. In Hinchey, M. and Coyle, L. (eds.), *Conquering Complexity*, pp. 277–307. Spinger.

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

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

[24] Burns, A. and Baxter, G. (2006) Time bands in systems structure. *Structure for Dependability*, pp. 74–90. Springer.

[25] Avizienis, A., Laprie, J.-C., Randell, B., and Landwehr, C. (2004) Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, **1**, 11–33.

[26] SAE (1996). Guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment, ARP 4761.

[27] Corsetti, E., Montanari, A., and Ratto, E. (1991) Dealing with different time granularities in formal specifications of real-time systems. *Journal of Real-Time Systems*, **3**, 191–215.

[28] Ciapessoni, E., Corsetti, E., Montanari, A., and Pietro, P. S. (1993) Embedding time granularity in a logical specification language for synchronous real-time systems. *Science of Computer Programming*, **20**, 141–171.

[29] Broy, M. and Stølen, K. (2001) *Specification and Development of Interactive Systems*. Springer-Verlag.

[30] Alur, R. (2015) *Principles of Cyber-Physical Systems*. MIT Press, Cambridge, Massachusetts.

[31] Chaochen, Z. and Hansen, M. R. (2004) *Duration Calculus*. Springer.

[32] Bjørner, D. (2006) *Software Engineering 3: Domains, Requirements, and Software Design*. Springer.

[33] Harel, D. and Politi, M. (1998) *Modeling Reactive Systems with Statecharts: The STATEMATE Approach*. McGraw-Hill.

[34] Parnas, D. L. and Madey, J. (1995) Functional documentation for computer systems engineering. *Sci. Comput. Program.*, **25**, 41–61.

[35] Dongol, B. and Hayes, I. J. (2012) Approximating idealised real-time specifications using time bands. *Automated Verification of Critical Systems 2011*, Electronic Communications of the EASST, **46**, pp. 1–16. EASST.

[36] Dongol, B. and Hayes, I. J. (2012) Rely/guarantee reasoning for teleo-reactive programs over multiple time bands. In

Derrick, J., Gnesi, S., Latella, D., and Treharne, H. (eds.), *Integrated Formal Methods*, LNCS, **7321**, pp. 39–53. Springer.

[37] Dongol, B., Hayes, I. J., and Derrick, J. (2014) Deriving real-time action systems with multiple time bands using algebraic reasoning. *Science of Computer Programming*, **85 Part B**, 137–165.

[38] Nilsson, N. (1994) Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research*, **1**, 139–158.

[39] Nilsson, N. (2001) Teleo-reactive programs and the triple-tower architecture. *Electronic Transactions on Artificial Intelligence*, **5**, 99–110.

[40] Hayes, I. J., Burns, A., Dongol, B., and Jones, C. B. (2013) Comparing degrees of non-determinstim in expression evaluation. *The Computer Journal*, **56**, 741–755.

[41] Mahony, B. P. and Hayes, I. J. (1992) A case-study in timed refinement: A mine pump. *IEEE Trans. on Software Engineering*, **18**, 817–826.

[42] Hayes, I. J. (2016) Generalised rely-guarantee concurrency: An algebraic foundation. *Formal Aspects of Computing*, **28**, 1057–1078.

[43] Dias, D. M. (2017) Mechanising an algebraic rely-guarantee refinement calculus. PhD thesis Newcastle University.