Journal of Universal Computer Science, vol. 6, no. 7 (2000), 704-730
submitted: 23/12/99, accepted: 20/4/00, appeared: 28/7/00 © Springer Pub. Co.

Incremental Development of Real-Time Requirements: The Light Control Case Study

Graeme Smith
Software Verification Research Centre,
University of Queensland, Australia
smith@svrc.uq.edu.au

Colin Fidge
Software Verification Research Centre,
University of Queensland, Australia
cjf@svrc.uq.edu.au

Abstract: System requirements frequently change while the system is still under development. Usually this means going back and revising the requirements specification and redoing those development steps already completed. In this article we show how formal requirements can be allowed to evolve while system development is in progress, without the need for costly redevelopment. This is done via a formalism which allows requirements engineering steps to be interleaved with formal development steps in a manageable way. The approach is demonstrated by a significant case study, the Light Control System.

Key Words: Requirements engineering; Formal specification; Refinement; Embedded systems; Real-time systems

Category: D.2.1 Requirements/Specifications; D.2.4 Software/Program Verification; J.7 Computers in Other Systems; C.3 Special-Purpose and Application-Based Systems

1 Introduction

In the literature, formal software development is usually described as a linear process in which an abstract requirements specification is 'refined', by a sequence of formal steps, to a concrete implementation [Morgan, 1990]. However, this process is predicated on the existence of a perfect requirements specification which anticipates all of the necessary system functionality, and allows for all the specific characteristics of the final implementation. In practice, however, requirements change during the development of a system and it is impossible to have prior knowledge of the final implementation technology. Programmers are therefore forced to undertake informal steps in which they revise their requirements specification, and redo much of their formal development to date, as the requirements are updated and the implementation becomes better defined.

Here we instead introduce a notion of 'realisation' which allows system requirements to evolve in a well-structured way as the formal development progresses [Smith, 2000]. The overall approach allows formal development (refinement) steps to be interleaved with requirements engineering (realisation) steps. As usual, the refinement steps allow a requirements specification to be transformed to a more specific design. By contrast, our realisation steps allow a specification or design to be changed, either by introducing new requirements or by modifying existing ones. Importantly, the realisation steps transform not only specifications, but also any formal properties already proven about those specifications. This allows formal reasoning carried out on the original specification to be reused after the specification has been modified, avoiding the need to repeat development steps and formal proofs.

We illustrate the approach by the incremental specification and development of part of the Light Control System, a significant case study involving both functional and timing requirements [Problem Description, 2000].

2 Previous and Related Work

The need to allow requirements specifications to evolve has been recognised for some time [Swartout and Balzer, 1982]. One reason for wanting this capability is that the practical limitations of implementation languages and hardware cannot usually be anticipated when the initial requirements document is written. This is particularly true of embedded real-time systems. Such systems interact with their environment via sensors and actuators, both of which introduce hardware-specific errors in timing (due to processing and propagation delays) and precision (due to the effects of quantization and bounded ranges).

Previously, this problem has been addressed by specifying the ideal behaviour of a real-time system independently from a description of the errors (precisions and tolerances) inherent in the environmental interface [van Schouwen et al., 1993]. This approach has been adopted in the U.S. Naval Research Laboratory's Software Cost Reduction methodology [Heitmeyer, 1996]. It has also been shown how to modify a specification in the Z notation to include physical limitations [Hayes, 1990] and, more generally, how to modify such specifications by changing their input and output representations [Hayes and Sanders, 1995]. In a broader context, this problem has also been addressed by the process of 'deidealizing' initial requirements by allowing them to be weakened [van Lamsweerde et al., 1995], and by the 'retrenchment' method of weakening specifications written in the B notation [Banach and Poppleton, 1998].

Other reasons why requirements must evolve is that they may initially be incomplete, may be subject to change from influences outside the project, or may be too complex to specify fully. Previously, these issues have been addressed by introducing notions of 'fundamental' versus 'changeable' requirements in order to manage the introduction of additional services into telephone networks [Bredereke, 1998]. Another approach is the notion of having 'default' assertions in specifications in order to capture properties which hold unless overridden [Guerra, 1999]. Both of these approaches assume some *a priori* knowledge of which parts of the specified functionality may change.

Our approach deals directly with all of these challenges [Smith, 2000]. In particular, it differs from all the previous work mentioned above in that it allows formal development steps to be arbitrarily interleaved with requirements engineering steps, and does not need any advance knowledge of which parts of the requirements specification are subject to change. Indeed, the approach is complete, in the sense that any changes can be made [Smith, 2000].

3 Definitions

This section briefly introduces the notations and rules used to develop the formal requirements for the Light Control System. We use a timed-trace specification notation, coupled with formal rules for refinement and realisation.

3.1 Specification Notation

Our specification notation is based on Mahony's timed refinement calculus, a formalism specifically designed for describing embedded real-time systems [Mahony and Hayes, 1992; Mahony, 1992; Fidge et al., 1998b].

Let $\mathbb{T}$ denote the time domain. All time-varying properties of the system are assumed to be represented by *timed traces*, i.e., functions from the time domain to the value of the variable at that time. For instance, given a variable property v capable of assuming values from set T, it is represented in the formalism as a function of type $\mathbb{T} \to T$. Below we assume that all variables have this underlying representation, and simply refer to v as having range type T.

Predicates on the system state are expressed using a set-theoretic notation in which the basic modelling unit is the time *interval*, i.e., a non-empty set of contiguous times [Fidge et al., 1998b]. In this article we use the notation $\langle P \rangle$ to denote the set of all intervals in which first-order predicate P is true at every time in the interval. (The interval endpoints may be either open or closed.) For conciseness, a timed-trace variable v may appear in predicate P without explicit indexing, in which case it is assumed to be indexed implicitly by all times in the interval. For instance, the set of intervals $\langle 0 < v \rangle$ comprises all intervals in which expression '$0 < v(t)$' is true at every time t in the interval. The reserved symbol δ may be used in predicate P to refer to the duration of the intervals.

Relationships between sets of time intervals are expressed using standard set operators such as $\cap$, $\cup$ and $\subseteq$. To simplify specification of requirements involving sequences of intervals, a concatenation operator is introduced for joining sets of intervals end-to-end. Given two sets of intervals $\langle P_1 \rangle$ and $\langle P_2 \rangle$, then '$\langle P_1 \rangle \,;\, \langle P_2 \rangle$' is the set of intervals formed by joining intervals from $\langle P_1 \rangle$ to intervals from $\langle P_2 \rangle$, provided that their endpoints meet at the same point in time [Fidge et al., 1998b].

Requirements are then expressed as *specification statements* of the form $\mathbf{x} : [A, E]$. This says that we are required to achieve the *effect* described by predicate E, by modifying only variables in the list $\mathbf{x}$. In doing so, we may make use of an *assumption* predicate A describing the expected behaviour of the inputs [Mahony, 1992]. If assumption A is merely 'true' then it can be omitted. Predicates A and E are constructed using the various time-interval notations described above.

Variables in the list $\mathbf{x}$ may be regarded as outputs of the system and may not occur free in A. All other variables occurring free in predicates A and E may be regarded as inputs. For a specification statement to be well-structured, it must be the case that E does not constrain any input variables [Mahony, 1992].

Several operators on specification statements are available. For instance, it is often useful to introduce *local variables* that are neither inputs or outputs. For example, variables $\mathbf{y}$ are local in the construct $\mathbf{x}, \mathbf{y} : [A, E] \setminus (\mathbf{y})$.

Three further operators can be used to compose specifications [Figure 1]. *Parallel* composition of two independent specifications S_1 and S_2 without communication is denoted $S_1 \mid S_2$. *Piping* composition, where outputs of specification S_1 are equated with identically-named inputs in S_2, is denoted $S_1 \gg S_2$ [Mahony, 1992]. We also introduce a *feedback* operator which allows local variables $\mathbf{x}$ to be used as both outputs and inputs, denoted $[\mu\ \mathbf{x}_0 \setminus \mathbf{x}] \bullet S$. Here specification S may define outputs $\mathbf{x}$ in its effect predicate, and may use assumptions about the corresponding inputs $\mathbf{x}_0$ in its assumption predicate. (Variables $\mathbf{x}_0$ are the *same* as $\mathbf{x}$, but are annotated differently to distinguish their role as inputs.)

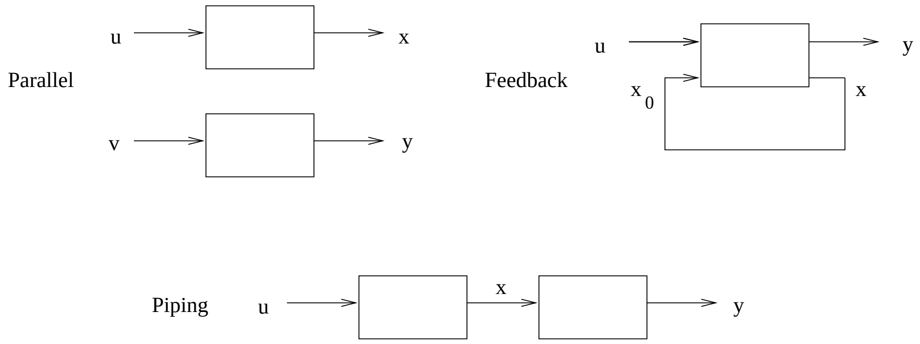

Figure 1: Compositional operators on specification statements

3.2 Refinement Rules

Refinement of a requirements specification formally transforms it by introducing implementation detail. The refinement relation $S_1 \sqsubseteq S_2$ says that specification S_1 is implemented by the more detailed specification S_2.

Two basic refinement rules for transforming the predicates within specification statements are as follows [Mahony, 1992].

Refinement Rule 1 (Weaken assumption)

$$\mathbf{x} : [A_1, E] \sqsubseteq \mathbf{x} : [A_2, E] \quad \text{provided } A_1 \Rightarrow A_2$$

Refinement Rule 2 (Strengthen effect)

$$\mathbf{x} : [A, E_1] \sqsubseteq \mathbf{x} : [A, E_2] \quad \text{provided } A \Rightarrow (\forall\, \mathbf{x} \bullet E_2 \Rightarrow E_1)$$

Notice that these rules may only be applied provided that the proof obligations on the right can be formally verified. (Refinement Rule 2 also applies to specifications without an explicit assumption A. Similarly for other such rules below.)

Refinement can also be used to introduce compositional operators. The rule for adding local variables to a specification is as follows [Mahony, 1992].

Refinement Rule 3 (Introduce local variables) *Assume that* $\mathbf{y} \cap \mathbf{x} = \varnothing$ *and variables* $\mathbf{y}$ *are not free in predicates* A *and* E.

$$\mathbf{x} : [A, E] \sqsubseteq \mathbf{x}, \mathbf{y} : [A, E] \setminus (\mathbf{y})$$

The effect predicate E can be subsequently strengthened to make use of the introduced variables $\mathbf{y}$. Conversely, a local variable can be removed if it is not used in any predicates.

Refinement Rule 4 (Remove local variables) *Assume that* $\mathbf{y} \cap \mathbf{x} = \varnothing$ *and variables* $\mathbf{y}$ *are not free in predicate* E.

$$\mathbf{x}, \mathbf{y} : [A, E] \setminus (\mathbf{y}) \sqsubseteq \mathbf{x} : [A, E]$$

Refinement rules for introducing the parallel, piping [Mahony, 1992] and feedback operators are given below. Let $P[\mathbf{x} \setminus \mathbf{y}]$ denote predicate P with all free occurrences of variables in list $\mathbf{x}$ replaced by the correspondingly positioned variables in list $\mathbf{y}$.

Refinement Rule 5 (Introduce parallelism) *Assume* $\mathbf{x} \cap \mathbf{y} = \varnothing$ *and variables* $\mathbf{x}$ *do not appear free in predicate* E_2 *and* $\mathbf{y}$ *do not appear free in* E_1.

$$\mathbf{x}, \mathbf{y} : [A, E_1 \wedge E_2] \sqsubseteq \mathbf{x} : [A, E_1] \mid \mathbf{y} : [A, E_2]$$

Refinement Rule 6 (Introduce piping) *Assume* $\mathbf{x} \cap \mathbf{y} = \varnothing$ *and variables* $\mathbf{y}$ *do not appear free in predicate* E_1.

$$\mathbf{y} : [A, E_1 \wedge E_2] \sqsubseteq \mathbf{x} : [A, E_1] \gg \mathbf{y} : [A \wedge E_1, E_2]$$

Refinement Rule 7 (Introduce feedback) *Assume variables* $\mathbf{x}_0$ *do not appear free in predicates* A *and* E_1.

$$\mathbf{x}, \mathbf{y} : [A, E_1] \setminus (\mathbf{x}) \sqsubseteq [\mu\ \mathbf{x}_0 \setminus \mathbf{x}] \bullet \mathbf{x}, \mathbf{y} : [A, E_2] \quad \text{provided } E_2[\mathbf{x}_0 \setminus \mathbf{x}] = E_1$$

3.3 Realisation Rules

Realisation of a requirements specification formally transforms it by introducing additional functionality or by changing existing functionality [Smith, 2000]. This is done in such a way that the properties of the original specification are transformed in a well-structured way. The satisfaction relation $S \vdash P$ indicates that specification S satisfies the properties defined by predicate P. As the realisation rules are less familiar than those for refinement, we also offer brief proofs of their correctness.

The general realisation rule for modifying the inputs and outputs of a specification is as follows. This rule, as well as transforming the specification, transforms all properties P of the specification as shown below. Let list $\mathbf{u}$ be the input and output variables being modified and F be a predicate relating $\mathbf{u}$ and the fresh local variables $\mathbf{u}_0$. If variables $\mathbf{u}$ are inputs then their modified values must satisfy the original specification's assumption A. This is because the ability to achieve the original specification's effect E may depend on this assumption. This is captured below by the proviso on predicates A and F.

Realisation Rule 1 (Modify effect) *Assume variables* $\mathbf{u}_0$ *do not appear free in predicates* A *and* E*.*

$$\begin{array}{rl} \text{If} & \mathbf{x} : [A, E] \vdash P \\ \text{then} & \mathbf{x}, \mathbf{u}_0 : [A, E[\mathbf{u} \setminus \mathbf{u}_0] \wedge F] \setminus (\mathbf{u}_0) \vdash \exists\, \mathbf{u}_0 \bullet P[\mathbf{u} \setminus \mathbf{u}_0] \wedge F \\ \text{provided} & A \Rightarrow (\forall\, \mathbf{u}_0 \bullet F \Rightarrow A[\mathbf{u} \setminus \mathbf{u}_0]) \end{array}$$

Proof:

$$\begin{array}{rl} & \mathbf{x} : [A, E] \vdash P \\ \equiv & \forall\, \mathbf{x} \bullet A \wedge E \Rightarrow P \\ \equiv & \forall\, \mathbf{x} \bullet (\exists\, \mathbf{u}_0 \bullet E[\mathbf{u} \setminus \mathbf{u}_0] \wedge A[\mathbf{u} \setminus \mathbf{u}_0]) \Rightarrow (\exists\, \mathbf{u}_0 \bullet P[\mathbf{u} \setminus \mathbf{u}_0]) \\ \equiv & \forall\, \mathbf{x} \bullet (\exists\, \mathbf{u}_0 \bullet E[\mathbf{u} \setminus \mathbf{u}_0] \wedge A[\mathbf{u} \setminus \mathbf{u}_0] \wedge F) \Rightarrow (\exists\, \mathbf{u}_0 \bullet P[\mathbf{u} \setminus \mathbf{u}_0] \wedge F) \\ \Rrightarrow & \text{`via the proviso on } F \text{ and } A\text{'} \\ & \forall\, \mathbf{x} \bullet A \wedge (\exists\, \mathbf{u}_0 \bullet E[\mathbf{u} \setminus \mathbf{u}_0] \wedge F) \Rightarrow (\exists\, \mathbf{u}_0 \bullet P[\mathbf{u} \setminus \mathbf{u}_0] \wedge F) \\ \equiv & \text{`via the semantics of local variables [Mahony, 1992]'} \\ & \mathbf{x}, \mathbf{u}_0 : [A, E[\mathbf{u} \setminus \mathbf{u}_0] \wedge F] \setminus (\mathbf{u}_0) \vdash \exists\, \mathbf{u}_0 \bullet P[\mathbf{u} \setminus \mathbf{u}_0] \wedge F \end{array}$$

□

This rule is monotonic. When the specification being modified is a component of a larger specification, and the other components of the larger specification do not refer to the variables being modified, then the properties of the larger specification are also transformed in the same way. The proof of this follows directly from the definitions of the compositional operators.

The following rule allows an assumption about the inputs to be added to a specification. It transforms the specification's properties as follows.

Realisation Rule 2 (Add assumption) *Assume variables* $\mathbf{x}$ *do not appear free in* A_2*.*

$$\begin{array}{rl} \text{If} & \mathbf{x} : [A_1, E] \vdash P \\ \text{then} & \mathbf{x} : [A_1 \wedge A_2, E] \vdash A_2 \Rightarrow P \end{array}$$

Proof:

$$\begin{array}{rl} & \mathbf{x} : [A_1, E] \vdash P \\ \equiv & \forall\, \mathbf{x} \bullet A_1 \wedge E \Rightarrow P \\ \Rrightarrow & \forall\, \mathbf{x} \bullet (A_1 \wedge A_2) \wedge E \Rightarrow (A_2 \Rightarrow P) \\ \equiv & \text{`via the semantics of specification statements [Mahony, 1992]'} \\ & \mathbf{x} : [A_1 \wedge A_2, E] \vdash A_2 \Rightarrow P \end{array}$$

□

As with Realisation Rule 1, this rule also extends to some systems of which the transformed specification is a component. In this case, the added assumption must not refer to variables which appear free in other components.

4 The Light Control System

The Light Control System is a challenge problem in system specification and requirements engineering [Problem Description, 2000]. It consists of an informal requirements document describing a planned control system for the lighting in a university building. The system is required to maintain a desired level of illumination in occupied rooms and hallways. In this article we formally develop the requirements for a single office. (Some slight simplifications are also made to the office description to avoid unnecessary repetition in the exposition. For instance, we do not distinguish between the two separate light fixtures in an office.)

The requirements document contains a natural-language requirements specification in the form of various system 'needs' [Problem Description, 2000]. These are presented from the perspective of either the 'user' (room occupant) or 'facility manager' (building superintendent). For instance, the user wants the lights to stay on if they leave the room for a short time, whereas the facility manager wants the lights to go off when a room is unoccupied to save energy. The specific needs relevant to the part of the case study we consider are described in detail below, as each is formalised [Sections 5.1–5.3]. The Light Control System requirements document is (deliberately) unclear in places, so we occasionally indicate design decisions made to resolve ambiguities.

The requirements document also describes selected details of the intended implementation. Each office has an external window and contains a ceiling light, motion detector and a control panel [Figure 2]. External light sensors are mounted outside the building. The room's occupant can select a desired level of illumination via the control panel. If nothing is chosen, there is a safe, default level of illumination. The motion detector enables the control system to detect whether the room is currently occupied. The outside light sensor measures the amount of daylight entering the office's window, and can thus help reduce energy costs by informing the control system when additional internal lighting is unnecessary. The control panel also contains a push button which can be used to manually switch the light on or off. To illustrate our ability to extend requirements, we initially ignore this feature and then introduce it towards the end of the paper. Certain physical characteristics of the sensors and actuators are described in the requirements document, and we also introduce these as our formal requirements evolve.

Omitted from this article are a number of needs concerning hallways and stairwells. We have also ignored some of the user-interface requirements for reporting malfunctions and energy consumption. There is no specific impediment to incorporating these features in our formalism, but it would be impossible to do them all justice in a single article.

5 Engineering the Light Control System Requirements

Our requirements engineering process interleaves both formal refinement and realisation steps [Figure 3]. Starting with the informal Light Control System requirements document, we produce an initial formal specification of the various system needs in the timed refinement calculus notation [Section 5.1]. This is then refined to introduce the motion detector and outside light sensor to the specification [Section 5.2.1]. However, at this level of abstraction, these components are

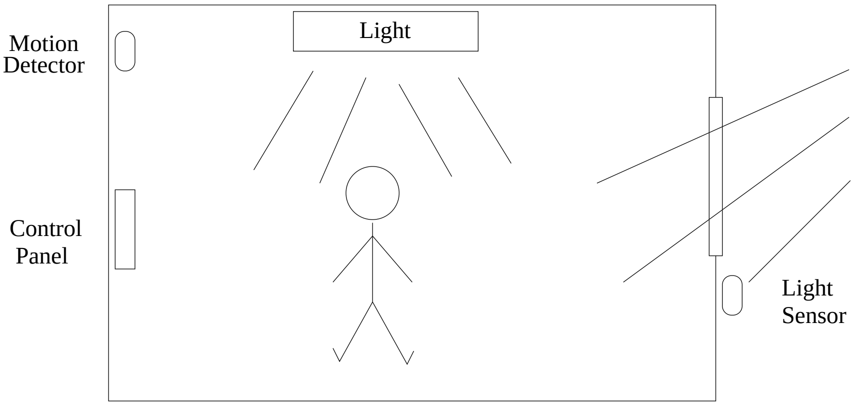

Figure 2: An office

described as 'ideal' devices, with no consideration given to their physical limitations. It would be impossible to further refine such specifications to a practical implementation. Therefore, we next undertake a realisation step, to relax the requirements on the speed and accuracy of the devices [Section 5.2.2]. A further refinement then instantiates the specific timing and precision characteristics provided for these devices in the Light Control System document [Section 5.2.3]. Up to this point, we ignore certain requirements entirely. In particular, there is a 'need' for the room's occupant to have access to a push-button override, and there are fault-tolerance requirements relating to failure of the sensors. Fortunately, our realisation steps allow these features to be introduced late in the design process [Sections 5.3.1 and 5.3.2]. Finally, further realisation and refinement steps introduce a feedback loop to the control system [Section 5.3.3], completing our formal design.

5.1 Overall Specification

Here we present an initial, abstract specification of the Light Control System, as elicited from the facility manager and user needs relevant to our simplified view of an office. We purposefully ignore certain requirements and implementation concerns in order to avoid the initial specification being too complicated, but will take advantage of the realisation process to introduce them later. This section describes the input and output variables, various predicates formally defining system requirements, and then collects these all together as a formal specification statement.

5.1.1 Variables and Constants

We begin by introducing some important system variables and constants. Let $\mathbb{B}$ be type Boolean, $\mathbb{N}$ the natural numbers, and $\mathbb{R}^+$ the non-negative reals.

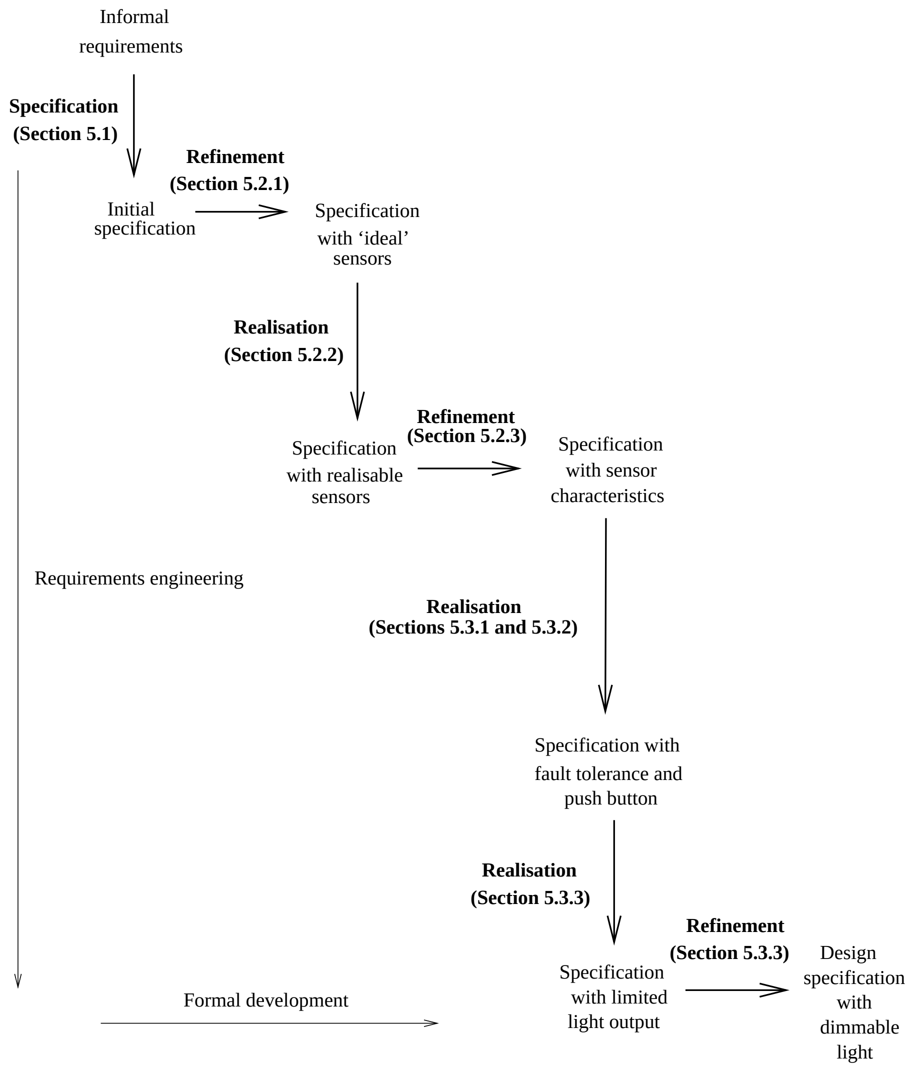

Figure 3: Summary of Light Control System requirements development

Three time-varying environmental properties form the overall inputs to our specification. Variable *person*, of range type $\mathbb{B}$, is true only when someone is in the room. Variable *chosen*, of range type $\mathbb{N}$, always equals the last illumination level (in lux) chosen by the room's occupant. If no level has ever been chosen then its value is arbitrary. Variable *daylight*, of range type $\mathbb{R}^+$, always equals the level of daylight illumination entering the window. (Recall that these variables are implicitly modelled as functions from the time domain [Section 3.1]. In particular, *daylight* is modelled as a differentiable function [Fidge et al., 1998a].)

Our top-level specification has only one observable output. Variable *light*, of range type $\mathbb{R}^+$, represents the illumination (in lux) being produced by the ceiling light.

We also introduce some (time invariant) constants. Constant *default*, of type $\mathbb{N}$, denotes the default value for the light when nothing has been chosen by the room's occupant. It must be greater than 14 lux, the minimum illumination required for safety [Problem Description, 2000]. Constants $t1$ and $t3$, of type $\mathbb{T}$, are non-negative durations (in seconds) required for timeouts associated with the Light Control System's behaviour.

Finally, we will use a local variable in the initial specification below. Let *desired*, of range type $\mathbb{N}$, represent the illumination level desired by the control system. (Under certain circumstances this may differ from the occupant's chosen illumination level.)

5.1.2 Facility Manager's Needs

The manager of the lighting in the building has certain requirements relating to energy efficiency. The first is captured by requirement FM1 of the Light Control System document [Problem Description, 2000]. When we reproduce such requirements here, we sometimes make minor changes either for our simplified office view or to remove ambiguity from the original requirement. Cases of the latter are indicated by italics.

> FM1: Use daylight to achieve the desired light setting of each *occupied* room whenever possible.

The italicised text in this case indicates that the notion of a 'desired' light setting is only relevant when the room is occupied. In particular, the light may be off when the room is unoccupied.

In our formalism, requirement FM1 is captured by the following predicate.

$$\begin{aligned} \mathit{FM}1 \stackrel{\text{def}}{=} \; & \langle \mathit{person} \wedge \mathit{daylight} \geqslant \mathit{desired} \rangle \subseteq \langle \mathit{light} = 0 \rangle \wedge \\ & \langle \mathit{person} \wedge \mathit{daylight} < \mathit{desired} \rangle \subseteq \langle \mathit{light} + \mathit{daylight} = \mathit{desired} \rangle \end{aligned}$$

The first conjunct states that if the room is occupied, and the level of daylight entering the window already meets the desired level, then the ceiling light should not be used. This is expressed by using the subset relation to require all time intervals in the left-hand set to also appear in the right-hand set. The second conjunct of $\mathit{FM}1$ states that if the room is occupied and the illumination from daylight is less than that desired, then the ceiling light should produce just enough additional illumination to meet the desired level.

The second facility manager need we consider concerns the requirement to conserve energy when a room is unoccupied [Problem Description, 2000].

> FM3: The ceiling light in a room has to be off when the room is unoccupied for at least T3 minutes.

This is captured by the following predicate. Let formal constant $t3$ be the number of seconds equivalent to T3 minutes.

$$FM3 \stackrel{\text{def}}{=} \langle \neg\, person \wedge \delta > t3 \rangle \subseteq \langle \text{true} \rangle \,;\, \langle light = 0 \rangle$$

This predicate states that if the room is unoccupied, then after T3 minutes the ceiling light must be off. On the left we have all intervals where the room is unoccupied, with a duration exceeding T3 minutes. All such intervals must also be in the set on the right, which is characterised by an initial, unspecified subinterval (property 'true'), followed by a subinterval in which the light is off.

5.1.3 User Needs

The occupant of a room also has particular needs. The first relates to their ability to choose the desired 'light scene' [Problem Description, 2000].

> U2: As long as the room is occupied, the chosen light scene has to be maintained.

This is formalised as follows.

$$\begin{aligned} U2 \stackrel{\text{def}}{=} \forall\, n : \mathbb{N} \bullet \langle person \rangle \cap (\langle chosen \neq n \rangle \,;\, \langle chosen = n \rangle) \\ \subseteq \langle \text{true} \rangle \,;\, \langle desired = n \rangle \end{aligned}$$

The predicate states that if the room is continuously occupied, and a new value n is chosen for the illumination, i.e., the value of *chosen* changes from some other value to n, then from that time the desired light setting must equal n. (All intervals in the left-hand set contain a point at which *chosen* changes. Furthermore, these intervals, which must also appear in the right-hand set, may be arbitrarily small. This constrains the point from which *desired* must equal n to be the same as the point where *chosen* changes.)

A subtle consequence of predicate $U2$ is that we do not recognise the act of *instantaneously* changing the chosen value from n to n. For such a 'change' to be perceptible to our specification, we assume that there must be *some* duration, no matter how short, during which the value of *chosen* takes some value other than n.

The next two needs define the light scene that must be established when someone first enters a room, depending on how long it has been left unoccupied.

> U3: If the room is reoccupied within T1 minutes after the last person has left the room, the previous light scene has to be reestablished *until a new light scene is chosen.*

We assume the value of the formal constant $t1$ is the number of seconds equivalent to T1 minutes.

$$\begin{aligned} U3 \stackrel{\text{def}}{=} \forall\, n, m : \mathbb{N} \bullet (\langle person \wedge desired = n \rangle \,;\, \langle \neg\, person \wedge \delta \leqslant t1 \rangle \,; \\ \langle person \wedge chosen = m \rangle) \\ \subseteq \langle \text{true} \rangle \,;\, \langle desired = n \rangle \end{aligned}$$

The predicate states that if the room is occupied with a desired illumination of n lux, and is then unoccupied for no more than T1 minutes, and is then reoccupied and a new value of *chosen* is not supplied (i.e., *chosen* remains constant), then the desired light scene must still be n. (Note that on reentering the room, the occupant's 'chosen' light scene, m, may be different from the 'desired' value n actually used. This situation can occur if the light scene when the person left the room was the 'default' one, rather than the 'chosen' one.)

> U4: If the room is reoccupied after more than T1 minutes since the last person has left the room, the default light scene has to be established *until a new light scene is chosen.*

This is formalised as follows.

$$\begin{aligned} U4 \overset{\text{def}}{=} \forall\, n : \mathbb{N} \bullet \langle \neg\, person \wedge \delta > t1 \rangle \,;\, \langle person \wedge chosen = n \rangle \\ \subseteq \langle \text{true} \rangle \,;\, \langle desired = default \rangle \end{aligned}$$

This predicate states that if the room has been unoccupied for more than T1 minutes, and after it is reoccupied the *chosen* value remains constant, then the desired light scene must be the default. (In the unlikely event that someone enters the room and chooses a new light scene at *exactly* the same instant, predicate $U4$ says that the default value still applies. The occupant must choose their new value some, possibly infinitesimal, time after reentering the room for it to have an effect.)

5.1.4 Initial Specification

Our initial specification of the overall Light Control System has three environmental inputs, *person*, *daylight* and *chosen*, and one output to the environment, *light* [Figure 4].

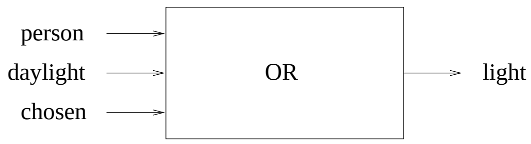

Figure 4: Top-level view of the Light Control System

The formal specification statement needs to ensure that all of the facility manager's and user's needs are met. This is done by conjoining all the needs defined above into a single Office Requirements predicate OR.

$$OR \overset{\text{def}}{=} FM1 \wedge FM3 \wedge U2 \wedge U3 \wedge U4$$

The specification of the whole system then uses OR as its effect predicate, and hides the local variable *desired*.

$$desired, light : [OR] \setminus (desired)$$

5.2 Physical Limitations

In this section, we show how physical limitations of the implementation technology can be introduced into the requirements specification. Firstly, we refine the overall system to introduce the motion detector and light sensor [Figure 5].

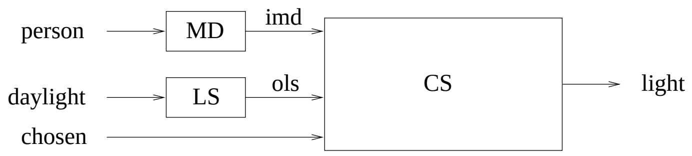

Figure 5: Control system and sensors

5.2.1 Refining the Initial Specification

We now transform the initial specification to a form closer to the intended implementation. The refinement introduces two new local variables: imd, of range type $\mathbb{B}$, represents the signal from the motion detector and ols, of range type $\mathbb{R}^+$, represents the signal from the light sensor.

The sensors are introduced as components which provide the control system with environmental information. In the refinement below we use the following Motion Detector MD and Light Sensor LS predicates.

$$MD \stackrel{\text{def}}{=} imd = person$$
$$LS \stackrel{\text{def}}{=} ols = daylight$$

These are ideal behaviours in which both sensors exactly match the properties they are tracking. For brevity in the refinement, we also introduce the following Control System predicate CS. It has the same behaviour as the previous office requirements, but substitutes the sensor outputs for the environmental properties they represent.

$$CS \stackrel{\text{def}}{=} OR[person, daylight \setminus imd, ols]$$

Refinement of the overall system requirements [Figure 4] to a design incorporating sensors [Figure 5] is done via the following formal steps.

$$\begin{array}{ll}
 & desired, light : [OR] \setminus (desired) \\
\sqsubseteq & \text{`by Refinement Rule 3, to introduce variables } imd \text{ and } ols\text{'} \\
 & imd, ols, desired, light : [OR] \setminus (desired, imd, ols) \\
\sqsubseteq & \text{`by Refinement Rule 2, to introduce predicates } MD \text{ and } LS\text{'} \\
 & imd, ols, desired, light : [MD \wedge LS \wedge OR] \setminus (desired, imd, ols)
\end{array}$$

$\sqsubseteq$ 'by Refinement Rule 6, to separate the sensors from the controller'
$(imd, ols : [MD \wedge LS] \gg desired, light : [MD \wedge LS, OR]) \setminus (desired)$
$\sqsubseteq$ 'by Refinement Rule 2, to use sensor outputs'
$(imd, ols : [MD \wedge LS] \gg desired, light : [MD \wedge LS, CS]) \setminus (desired)$
$\sqsubseteq$ 'by Refinement Rule 1, to eliminate unnecessary assumption'
$(imd, ols : [MD \wedge LS] \gg desired, light : [CS]) \setminus (desired)$
$\sqsubseteq$ 'by Refinement Rule 5, to separate the two sensors'
$((imd : [MD] \mid ols : [LS]) \gg desired, light : [CS]) \setminus (desired)$

Usually, the next step in a formal development of this specification would be to refine the sensors to more closely match their final implementations. However, in practice, such devices are inevitably limited in their reaction and data-conversion speeds, and the resolution and range of values they can represent. Therefore, a formal refinement of the sensor specifications above is not possible because these physical limitations have not been allowed for. For example, predicate *MD* states that the motion detector *instantaneously* reports the presence of a person in the room. No implementation could ever satisfy this requirement. Fortunately, our realisation laws offer a way forward, rather than going back to revise the development so far.

5.2.2 Realising the Sensors

Here we show how the 'ideal' sensor specifications introduced above can be transformed to realisable (i.e., implementable) ones. This is done using our realisation rules [Section 3.3] which also transform established properties already proven about the initial system specification.

5.2.2.1 Motion Detector

In practice, a motion detector has a finite response time. Therefore, its output cannot equal variable *person* (as in the ideal case) but is a 'delayed' version of this variable. We use function *delay* to model the effect of delaying a signal by a duration d [Figure 6].

$$delay = \lambda\, d : \mathbb{T} \bullet (\lambda\, t : \mathbb{T} \bullet t - d)$$

We can now transform our ideal motion detector, i.e., $imd : [MD]$, to one that has some arbitrary delay d. We define the following predicate in which a fresh variable $person_{del}$, of range type $\mathbb{B}$, is used to represent a delayed version of function *person*.

$$MD_{del} \stackrel{\text{def}}{=} MD[person \setminus person_{del}] \wedge (\exists\, d : \mathbb{T} \bullet person_{del} = person \circ delay(d))$$

We then apply Realisation Rule 1 to $imd : [MD]$ to produce the following specification.

$$imd, person_{del} : [MD_{del}] \setminus (person_{del})$$

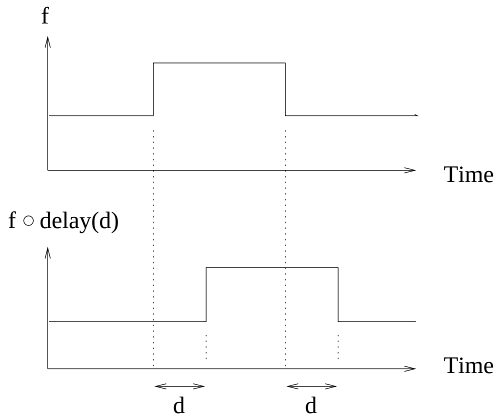

Figure 6: Effect of *delay* on a function f

The initial system specification [Section 5.1] satisfied several formal properties such as $U4$, the requirement to (instantaneously) establish default lighting when someone enters the room after it has been left unoccupied for a significant period. However, after the above realisation step, our system no longer satisfies this property because it may take some time to detect that someone has entered the room. Nevertheless, Realisation Rule 1 allows us to transform $U4$ so that the following corresponding property of the new specification can be discovered.

$$
\begin{array}{l}
\quad \exists\, person_{del} : \mathbb{T} \rightarrow \mathbb{B} \bullet \\
\qquad U4[person \setminus person_{del}] \wedge (\exists\, d : \mathbb{T} \bullet person_{del} = person \circ delay(d)) \\
\equiv \exists\, person_{del} : \mathbb{T} \rightarrow \mathbb{B} \bullet \\
\qquad (\forall\, n : \mathbb{N} \bullet \\
\qquad\quad \langle \neg\, person_{del} \wedge \delta > t1 \rangle \,;\, \langle person_{del} \wedge chosen = n \rangle \\
\qquad\qquad \subseteq \langle \mathrm{true} \rangle \,;\, \langle desired = default \rangle) \wedge \\
\qquad (\exists\, d : \mathbb{T} \bullet person_{del} = person \circ delay(d)) \\
\equiv \exists\, d : \mathbb{T} \bullet \\
\qquad (\forall\, n : \mathbb{N} \bullet \\
\qquad\quad \langle \neg\, person \circ delay(d) \wedge \delta > t1 \rangle \,;\, \langle person \circ delay(d) \wedge chosen = n \rangle \\
\qquad\qquad \subseteq \langle \mathrm{true} \rangle \,;\, \langle desired = default \rangle) \\
\equiv \exists\, d : \mathbb{T} \bullet \\
\qquad (\forall\, n : \mathbb{N} \bullet \\
\qquad\quad \langle \neg\, person \wedge \delta > t1 \rangle \,;\, \langle person \wedge \delta = d \rangle \,;\, \langle person \wedge chosen = n \rangle \\
\qquad\qquad \subseteq \langle \mathrm{true} \rangle \,;\, \langle desired = default \rangle)
\end{array}
$$

Therefore, in the new specification, default lighting is only guaranteed to be established after the room has been occupied for d seconds. This fact was derived straightforwardly from property $U4$ which was known to hold for the initial specification. In general, this approach is more efficient than deriving such properties from the new specification alone.

5.2.2.2 Light Sensor

As well as a response delay, a practical light sensor is limited in the resolution and range of values it can generate. In our case this can be modelled by associating an error range with the input variable *daylight*. We define the following predicate LS_{err} to represent the behaviour of the light sensor with an error up to e lux. Here fresh variable $daylight_{err}$, of range type $\mathbb{R}^+$, denotes the perceived, rather than actual, level of daylight.

$$\begin{aligned} LS_{err} \stackrel{\text{def}}{=} & LS[daylight \setminus daylight_{err}] \wedge \\ & (\exists\, e : \mathbb{R}^+ \bullet (\forall\, t : \mathbb{T} \bullet daylight_{err}(t) \in daylight(t) \pm e)) \end{aligned}$$

Ideal sensor requirement $ols : [LS]$ is then transformed to the following specification using Realisation Rule 1.

$$ols, daylight_{err} : [LS_{err}] \setminus (daylight_{err})$$

Again, the realisation rule allows us to derive properties of the new specification from the old one. For example, the following predicate can be deduced from requirement $FM1$ and hence from the initial specification.

$$\langle person \wedge daylight < desired \rangle \subseteq \langle light + daylight = desired \rangle$$

It says that when the room is occupied, and there is insufficient daylight, then the ceiling light must make up the shortfall. From this, and Realisation Rule 1, we can determine that the following is a property of the new specification.

$$\begin{aligned} & \exists\, daylight : \mathbb{T} \rightsquigarrow \mathbb{R}^+ \bullet \\ & \quad \langle person \wedge daylight_{err} < desired \rangle \subseteq \langle light + daylight_{err} = desired \rangle \wedge \\ & \quad (\exists\, e : \mathbb{R}^+ \bullet (\forall\, t : \mathbb{T} \bullet daylight_{err}(t) = daylight(t) \pm e)) \\ \equiv\ & \langle person \wedge daylight + e < desired \rangle \subseteq \langle light + daylight \in desired \pm e \rangle \end{aligned}$$

In other words, the ceiling light is only guaranteed to be used if the actual level of daylight falls short of the desired level by at least the error margin e. Also, the actual illumination in the room may vary from the desired value by up to e lux.

In practice, to produce an output which is guaranteed to always remain within a certain range of an input, it is necessary to have some knowledge about the input's maximum rate of change. We would therefore like to add an assumption about the rate of change of *daylight* to our light sensor specification. This can be done via a further transformation using Realisation Rule 2. This assumption can also capture a bound on the input range. Let constant R, of type $\mathbb{R}^+$, be the maximum rate of change of daylight (in lux/second), and constant B, of type $\mathbb{N}$, be the maximum brightness measurable by the light sensor (in lux). We define a Daylight Characteristics predicate DC to capture the assumptions. Let f' be the derivative of function f, and $|\cdot|$ denote magnitude.

$$DC \stackrel{\text{def}}{=} \forall\, t : \mathbb{T} \bullet |daylight'(t)| \leqslant R \wedge daylight(t) \leqslant B$$

Using Realisation Rule 2, the light sensor requirements can then be transformed to the following specification.

$$ols, daylight_{err} : [DC, LS_{err}] \setminus (daylight_{err})$$

Realisation Rule 2 also tells us that the property proven above about the light sensor specification with bounded error can be further transformed as follows.

$$(\forall t : \mathbb{T} \bullet |daylight'(t)| \leqslant R \wedge daylight(t) \leqslant B) \Rightarrow$$
$$\langle person \wedge daylight + e < desired \rangle \subseteq \langle light + daylight \in desired \pm e \rangle$$

That is, the requirement to produce an acceptable value for $light$, within the error range e, holds only if $daylight$ changes no faster than R and does not exceed B.

5.2.3 Refining the Sensors

We can now formally refine our sensor specifications to include the implementation-specific device characteristics provided by the Light Control System requirements document [Table 1].

Sensor	Resolution	Range	Reaction Time	Conversion Time
motion detector			1s	
light sensor	1 lux	1-10000 lux	10ms	1s

Table 1: Sensor characteristics [Problem Description, 2000]

5.2.3.1 Motion Detector

We define a specific motion detector predicate MD_1 capturing a 1 second delay between $person$ and imd as follows.

$$MD_1 \stackrel{\text{def}}{=} imd = person \circ delay(1)$$

The motion detector specification [Section 5.2.2.1] is then refined straightforwardly to include this specific lag in responsiveness.

$$imd, person_{del} : [MD_{del}] \setminus (person_{del})$$
$$\sqsubseteq \text{‘by Refinement Rules 2 and 4, instantiating } d \text{ as 1’}$$
$$imd : [MD_1]$$

5.2.3.2 Light Sensor

Our aim here is to refine the light sensor specification [Section 5.2.2.2] to a design which periodically samples the outside light level. We first introduce a function *sample* which, given a duration T representing the sampling period, maps all times t to the most recent whole multiple of T [Figure 7]. Let 'div' denote integer (truncating) division.

$$sample = \lambda\, T : \mathbb{T} \bullet (\lambda\, t : \mathbb{T} \bullet (t \operatorname{div} T) * T)$$

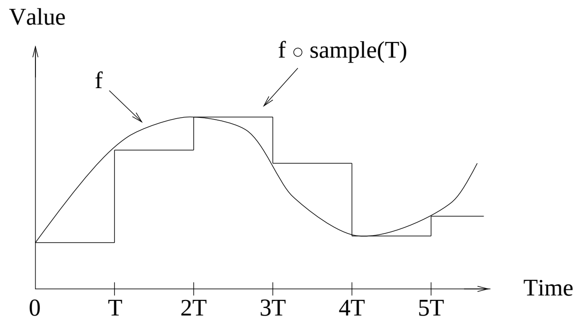

Figure 7: Effect of *sample* on a function f

The actual light sensor has a 10 millisecond reaction time [Table 1] and so, rather than sampling at time T, will sample at time $T + 0.01$. Furthermore, due to its conversion delay, the sampled value is not available until another second has passed, i.e., at time $T + 1.01$. This sampled value may also have up to 1 lux error due to the sensor's limited resolution [Figure 8].

Our earlier light sensor specification [Section 5.2.2.2] required that the output remains within e lux of the input [Figure 8]. In general, this property will be satisfied provided the value sampled at time $T + 0.01$ is still within e lux of the actual daylight value $T + 1$ seconds later, i.e., at time $2T + 1.01$, when the next sample becomes available. Since the sample at time $T + 0.01$ may err by as much as 1 lux, the level of daylight must change by no more than $e - 1$ lux in $T + 1$ seconds. Therefore, given that R is the maximum rate of change of the daylight, we require the property that $R \leqslant (e-1)/(T+1)$ lux/second. To achieve this, we must select a sampling period T such that $T \leqslant (e-1)/R - 1$.

We define the following predicate LS_T to model a light sensor with sampling period T. A fresh variable $daylight_{res}$, of range type $\mathbb{R}^+$, represents a value that is always within 1 lux of *daylight*, to account for the resolution error. This variable is sampled using the *sample* function delayed by 10 milliseconds and then the whole signal is delayed a further 1 second to model the light sensor's conversion delay.

$$\begin{aligned} LS_T \stackrel{\text{def}}{=} {} & (\forall\, t : \mathbb{T} \bullet daylight_{res}(t) \in daylight(t) \pm 1) \wedge \\ & ols = (daylight_{res} \circ (sample(T) \circ delay(0.01))) \circ delay(1) \end{aligned}$$

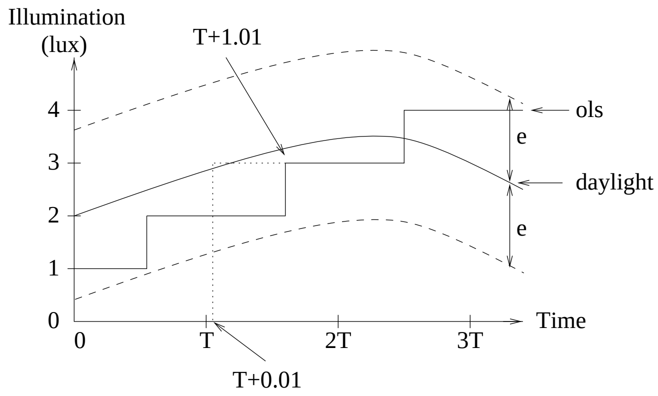

Figure 8: Light sensor input ($daylight$) versus output (ols)

Our previous light sensor specification [Section 5.2.2.2] can then be refined as follows.

$$
\begin{array}{l}
\quad ols, daylight_{err} : [DC, LS_{err}] \setminus (daylight_{err}) \\
\sqsubseteq \text{`by Refinement Rule 3, to introduce local variable } daylight_{res}\text{'} \\
\quad ols, daylight_{err}, daylight_{res} : [DC, LS_{err}] \setminus (daylight_{err}, daylight_{res}) \\
\sqsubseteq \text{`by Refinement Rules 2 and 4'} \\
\quad ols, daylight_{res} : [DC, LS_T] \setminus (daylight_{res})
\end{array}
$$

The proof obligation associated with the application of Refinement Rule 2 can be proven correct provided that the value chosen for period T is related to R and e as explained above.

5.3 New Requirements

Here we show how the realisation rules can also be used to deal with new requirements. In our development so far we have ignored the requirements for fault tolerance and the push button for the ceiling light. We now introduce these requirements and refine our specification to a particular target architecture, which includes a 'dimmable light' component, described in the Light Control System requirements document [Problem Description, 2000].

5.3.1 Fault Tolerance

In response to sensor failure, the control system must provide a stepwise degradation of its functionality. Two requirements associated with this fault tolerant behaviour are relevant to our simplified view: NF1 and NF4 [Problem Description, 2000].

> NF1: If any outdoor light sensor does not work correctly, the control system for rooms should behave as if the outdoor light sensor had been submitting the last correct measurement of the outdoor light constantly.

Our goal is to add this functionality to the control system component of our specification, i.e., $desired, light : [CS]$ [Section 5.2.1]. To do this we introduce an additional auxiliary input variable ols_status, of range type $\mathbb{B}$, to denote the status (working or failed) of the light sensor, and a local variable ols_{ft}, of range type $\mathbb{R}^+$, to denote the value that the fault-tolerant control system uses in place of the value from the light sensor. The additional requirement is then specified by the following predicate.

$$\begin{aligned} NF1 \stackrel{\text{def}}{=} & (\forall\, r : \mathbb{R}^+ \bullet \langle ols\_status \wedge ols = r\rangle \,;\, \langle \neg\, ols\_status\rangle \subseteq \langle ols_{ft} = r\rangle) \wedge \\ & \langle ols\_status\rangle \subseteq \langle ols_{ft} = ols\rangle \end{aligned}$$

The first conjunct models the situation when the sensor fails, that is, when ols_status changes from 'true' to 'false'. In this case ols_{ft} stays equal to r, the last 'good' value of ols before the failure. The second conjunct models the situation when the sensor has not failed, in which case ols_{ft} equals the actual value of ols.

> NF4: If any motion detector of a room does not work correctly, the control system should behave as if the room were occupied.

This requirement is also added to the control system component of our specification. We introduce an additional input imd_status, of range type $\mathbb{B}$, to represent the status of the motion detector, and a local variable imd_{ft}, of range type $\mathbb{B}$, to denote the value that the fault-tolerant control system uses in place of the value from the motion detector. The additional requirement is specified by the following predicate.

$$\begin{aligned} NF4 \stackrel{\text{def}}{=} & \langle \neg\, imd\_status\rangle \subseteq \langle imd_{ft}\rangle \wedge \\ & \langle imd\_status\rangle \subseteq \langle imd_{ft} = imd\rangle \end{aligned}$$

The first conjunct deals with the situation where the motion detector has failed, i.e., when imd_status is false, in which case imd_{ft} must equal 'true' to indicate that the room is occupied. The second conjunct models the situation where the motion detector is working, in which case imd_{ft} equals the actual value of imd.

To add these requirements to the control system specification, we use Realisation Rule 1 to conjoin the new requirements to the effect predicate, and to introduce the new local variables imd_{ft} and ols_{ft}. Since we require the control system to use these local variables in place of its original imd and ols inputs, we define the following Fault-Tolerant Control System predicate CS_{ft}.

$$CS_{ft} \stackrel{\text{def}}{=} CS[imd, ols \setminus imd_{ft}, ols_{ft}]$$

The revised control system specification resulting from application of Realisation Rule 1 is then as follows.

$$desired, light, ols_{ft}, imd_{ft} : [CS_{ft} \wedge NF1 \wedge NF4] \setminus (imd_{ft}, ols_{ft})$$

5.3.2 Push Button

An office must have a push button which enables the ceiling light to be turned off, overriding the control system [Problem Description, 2000]. The effect of pushing the button is described in the following text from the requirements document which we label PB.

> PB: A ceiling light in an *occupied* room shows the following behaviour when the corresponding push button is pushed:
> (i) if the ceiling light is on, it will be switched off
> (ii) otherwise it will be switched on.
> *There is no other way to switch the ceiling light in an occupied room on or off.*

To add this requirement to our control system specification, we introduce a new input *button*, of range type $\mathbb{B}$. Since pushing the button toggles the system state, we let the transitions of this variable (from 'true' to 'false' or 'false' to 'true') represent the instants when the button is pushed. Later, this particular abstract representation could be refined to a more concrete one where, for example, the state change is triggered by the rising edge of a voltage signal indicating closure of a circuit [Figure 9].

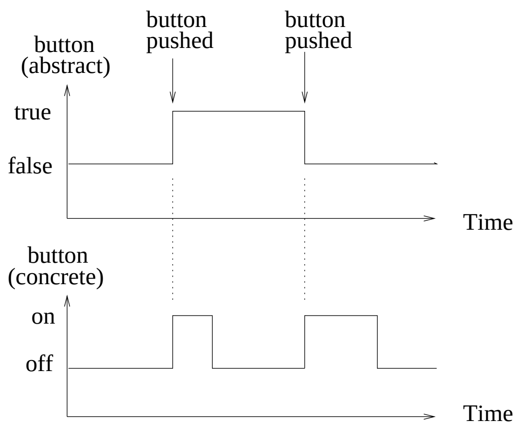

Figure 9: A possible refinement of *button*

We also introduce two local variables. Variable *light_on*, of range type $\mathbb{B}$, denotes whether the light is switched on according to the push button. So far, the control system component has directly produced output *light*. So that this output can now be overridden by the push button, we instead introduce a local variable $light_{req}$, of range type $\mathbb{R}^+$, which denotes the level of illumination requested by the control system. The additional requirement is then captured by the following

Push Button predicate PB.

$$\begin{aligned} PB \stackrel{\text{def}}{=} & \langle person \rangle \cap (\langle \neg\ button \rangle ; \langle button \rangle \cup \langle button \rangle ; \langle \neg\ button \rangle) \\ & \quad = \langle light\_on \rangle ; \langle \neg\ light\_on \rangle \cup \langle \neg\ light\_on \rangle ; \langle light\_on \rangle \wedge \\ & \langle \neg\ light\_on \rangle \subseteq \langle light = 0 \rangle \wedge \\ & \langle light\_on \rangle \subseteq \langle light = light_{req} \rangle \end{aligned}$$

The first conjunct states that if the room is occupied and variable $button$ changes value, then variable $light\_on$ also changes value. The second conjunct states that when $light\_on$ is 'false' then the ceiling light must be off (regardless of the value of the control system's $light_{req}$ output). The third conjunct states that when $light\_on$ is 'true', the external output $light$ equals the value of variable $light_{req}$ generated by the control system.

Since the fault-tolerant control system will now produce an internal value $light_{req}$, rather than directly generating $light$, we define the following Control System Request predicate CS_{req} which substitutes output $light_{req}$ for $light$.

$$CS_{req} \stackrel{\text{def}}{=} (CS_{ft} \wedge NF1 \wedge NF4)[light \setminus light_{req}]$$

The push button requirement PB is then conjoined to the control system specification using Realisation Rule 1 (preceded by Refinement Rule 3 to introduce $light\_on$), resulting in the following specification.

$$\begin{aligned} & desired, light, ols_{ft}, imd_{ft}, light_{req}, light\_on : \\ & \quad [CS_{req} \wedge PB] \setminus (imd_{ft}, ols_{ft}, light_{req}, light\_on) \end{aligned}$$

5.3.3 Design Specification with Dimmable Light

The Light Control System requirements propose an internal architecture for the overall system which includes a Light Controller component LC and a Dimmable Light component DL [Problem Description, 2000]. A feedback loop enables the controller to tell whether the light is on or off [Figure 10].

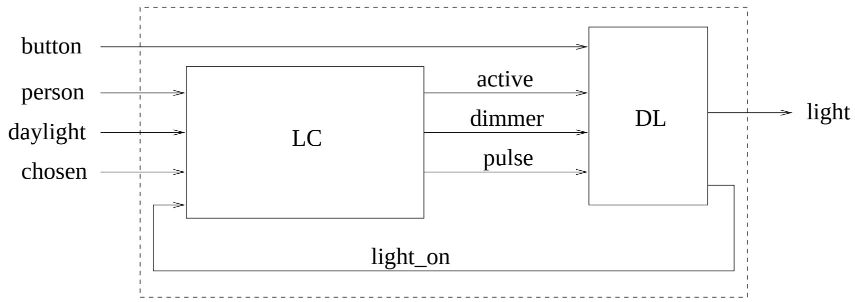

Figure 10: Structure of system including dimmable light

The behaviour of the dimmable light is described in the following text from the requirements document [Problem Description, 2000] which we label as requirement DL1.

> DL1: Inputs to a dimmable light are created by a pulse to toggle the light, by a dimmer to set the current dim value, and by *a* control system active *signal* to show the status of the control system. If this signal is not sent every 60s, the dimmable light switches to fail safe mode, i.e., dim value is assumed to be 100%.

In order to introduce this structure to our specification, we need to model this fail safe mode and, therefore, must define what is meant by '100%' illumination. Our specification so far has placed no bound on the level of light requested by the user or supplied by the ceiling light. This is clearly unimplementable, so we must perform another realisation step before our refinement. Let constant Max, of type $\mathbb{N}$, be the maximum illumination (in lux) of the ceiling light. The following Bounded Choice predicate BC then places a limit on the chosen illumination value.

$$BC \stackrel{\text{def}}{=} chosen \leqslant Max$$

Using Realisation Rule 2, we add this predicate as an assumption to the control system specification above [Section 5.3.2] as follows.

$$\begin{aligned} &desired, light, ols_{ft}, imd_{ft}, light_{req}, light\_on : \\ &\qquad [BC, CS_{req} \wedge PB] \setminus (imd_{ft}, ols_{ft}, light_{req}, light\_on) \end{aligned}$$

In other words, the control system is only required to produce the desired level of illumination provided that the user does not request more light than the system can generate.

Requirement DL1 introduces three signals which must be sent between the controller and the dimmable light. Variable *dimmer*, of range type $\mathbb{R}^+$, represents the level of illumination (in lux) requested of the dimmable light by the controller. Variable *pulse*, of range type $\mathbb{B}$, is used by the controller to switch the light on or off. It is interpreted in the same manner as *button*, i.e., a transition (from 'true' to 'false' or 'false' to 'true') is deemed to represent the instant a pulse occurs. Variable *active*, also of range type $\mathbb{B}$, is used by the controller to indicate that it is working. It is 'false' except at single instants of time corresponding to the sending of the active signal. Again, these two signal abstractions could be refined to match a number of different hardware implementations [Figure 11].

Informal requirement DL1 is formalised by the following predicate.

$$\begin{aligned} DL1 \stackrel{\text{def}}{=} &\langle \neg\, pulse \rangle \,;\, \langle pulse \rangle \cup \langle pulse \rangle \,;\, \langle \neg\, pulse \rangle \\ &\qquad \subseteq \langle light\_on \rangle \,;\, \langle \neg\, light\_on \rangle \cup \langle \neg\, light\_on \rangle \,;\, \langle light\_on \rangle \wedge \\ &\langle light\_on \wedge active \rangle \,;\, \langle light\_on \wedge \delta \leqslant 60 \rangle \subseteq \langle light = dimmer \rangle \wedge \\ &\langle light\_on \wedge \neg\, active \wedge \delta > 60 \rangle \subseteq \langle \text{true} \rangle \,;\, \langle light = Max \rangle \end{aligned}$$

The first conjunct states that transitions of controller signal *pulse* must toggle the dimmable light's state. The second says that the *light* output will equal the

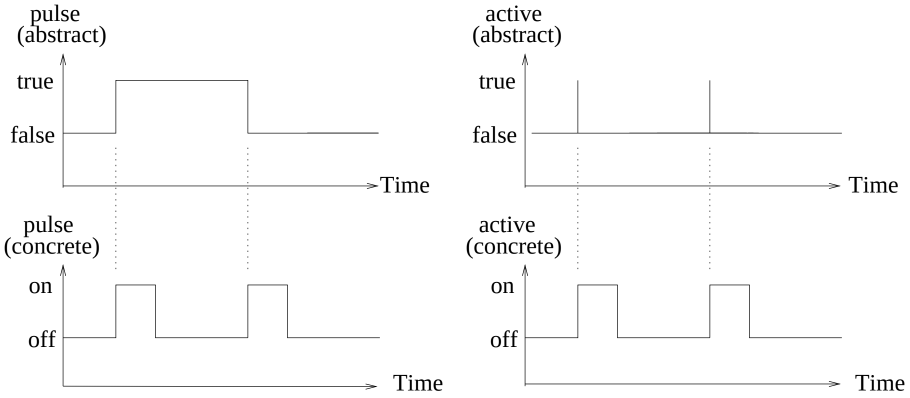

Figure 11: Possible refinements of *pulse* and *active*

value requested by variable *dimmer*, provided the controller has indicated that it is active within the last 60 seconds. The third conjunct says that at the end of any interval exceeding 60 seconds, where no *active* signal has been produced by the controller, the *light* output should equal its maximum value.

An additional controller requirement (not explicitly stated in the Light Control System requirements document) is needed to ensure that requirements FM3, U3 and U4 are met by the restructured system. We label this requirement DL2.

> DL2: *The pulse is triggered when the light is off and a person enters an unoccupied room, and when the light is on and T3 minutes has expired with the room being unoccupied.*

This is formalised by the following predicate.

$$\begin{aligned} DL2 \stackrel{\text{def}}{=} & \langle \neg\, person \wedge \neg\, light\_on \rangle \,;\, \langle person \rangle \\ & \quad \subseteq \langle \neg\, pulse \rangle \,;\, \langle pulse \rangle \cup \langle pulse \rangle \,;\, \langle \neg\, pulse \rangle \wedge \\ & \langle person \rangle \,;\, (\langle \neg\, person \wedge \delta \geqslant t3 \rangle \cap (\langle \text{true} \rangle \,;\, \langle light\_on \rangle)) \\ & \quad \subseteq \langle \delta = t3 \rangle \,;\, (\langle \neg\, pulse \rangle \,;\, \langle pulse \rangle \cup \langle pulse \rangle \,;\, \langle \neg\, pulse \rangle) \end{aligned}$$

The first conjunct captures the triggering of *pulse* whenever a person enters an unoccupied room. The second captures the triggering of a pulse T3 minutes after a room has been unoccupied, provided the light is on.

We use the following predicates in the refinement. Firstly, a Light Controller predicate LC consists of the control system properties already defined above, but with *dimmer* as output, rather than $light_{req}$. It also has the new requirement DL2 with sensor inputs imd_{ft} and ols_{ft} substituting for values from the environment.

$$LC \stackrel{\text{def}}{=} CS_{req}[light_{req} \setminus dimmer] \wedge DL2[person, daylight \setminus imd_{ft}, ols_{ft}]$$

Secondly, a Dimmable Light predicate DL consists of the push-button properties described above, but with $dimmer$ as its input instead of $light_{req}$, plus the new requirements defined by predicate $DL1$.

$$DL \stackrel{\text{def}}{=} PB[light_{req} \setminus dimmer] \land DL1$$

Refinement to the suggested system architecture [Figure 11] then proceeds via the following formal steps.

$$
\begin{array}{l}
\quad desired, light, ols_{ft}, imd_{ft}, light_{req}, light\_on : \\
\qquad [BC, CS_{req} \land PB] \setminus (imd_{ft}, ols_{ft}, light_{req}, light\_on) \\
\sqsubseteq \text{`by Refinement Rule 3, to introduce } pulse, dimmer \text{ and } active\text{'} \\
\quad desired, light, ols_{ft}, imd_{ft}, light_{req}, light\_on, pulse, dimmer, active : \\
\qquad [BC, CS_{req} \land PB] \setminus (imd_{ft}, ols_{ft}, light_{req}, light\_on, pulse, dimmer, active) \\
\sqsubseteq \text{`by Refinement Rule 2, given property DL2'} \\
\quad desired, light, ols_{ft}, imd_{ft}, light\_on, pulse, dimmer, active : \\
\qquad [BC, LC \land DL] \setminus (imd_{ft}, ols_{ft}, light\_on, pulse, dimmer, active) \\
\sqsubseteq \text{`by Refinement Rule 7, to feedback } light\_on\text{'} \\
\quad [\mu \;\; light\_on_0 \setminus light\_on] \bullet \\
\qquad desired, light, ols_{ft}, imd_{ft}, light\_on, pulse, dimmer, active : \\
\qquad\quad [BC, LC[light\_on \setminus light\_on_0] \land DL] \\
\qquad\quad \setminus (imd_{ft}, ols_{ft}, pulse, dimmer, active) \\
\sqsubseteq \text{`by Refinement Rule 6, to separate controller and light'} \\
\quad [\mu \;\; light\_on_0 \setminus light\_on] \bullet \\
\qquad (desired, ols_{ft}, imd_{ft}, pulse, dimmer, active : \\
\qquad\quad [BC, LC[light\_on \setminus light\_on_0]] \gg \\
\qquad\quad light, light\_on : [BC \land LC[light\_on \setminus light\_on_0], DL]) \\
\qquad\quad \setminus (imd_{ft}, ols_{ft}) \\
\sqsubseteq \text{`by Refinement Rule 1, to remove assumption from dimmable light'} \\
\quad [\mu \;\; light\_on_0 \setminus light\_on] \bullet \\
\qquad (desired, ols_{ft}, imd_{ft}, pulse, dimmer, active : \\
\qquad\quad [BC, LC[light\_on \setminus light\_on_0]] \gg light, light\_on : [DL]) \\
\qquad\quad \setminus (imd_{ft}, ols_{ft})
\end{array}
$$

This completes our overall development process. Of course, the principles of refinement and realisation would allow us to take this process further, introducing, for example, jitters in the perfect delays we have specified, however this goes beyond the information in the requirements document [Problem Description, 2000]. Starting from an informal set of natural-language requirements, we have incrementally developed a detailed, precise formal specification. While this specification models only a single office, we fully expect the approach to scale up to a specification comprising multiple offices using conjunction to combine parameterised, but otherwise identical, office specifications. Similarly, hallways and other types of rooms could be added. Component specifications could also ultimately be transformed to code, where appropriate, using the sequential real-time refinement calculus of Hayes and Utting [Hayes and Utting, 1997].

6 Conclusion

We have shown, via a significant case study, how requirements engineering and formal development steps may coexist. The approach avoids the need to redo development steps due to changes in requirements, yet retains all the advantages of a verifiable, auditable formal development process. Although the final design does not correspond exactly to the initial requirements formally specified, there is a complete documented path between these requirements and the design. Tool support for the notation is currently being developed [Cerone, 2000].

Acknowledgements

This research was supported by Australian Research Council Large Grant A49801500, *A Unified Formalism for Concurrent Real-Time Software Development*.

References

[Banach and Poppleton, 1998] Banach, R. and Poppleton, M. Retrenchment: An engineering variation on refinement. In Bert, D., editor, *Proceedings of B-98*, volume 1393 of *LNCS*, pages 129–147. Springer-Verlag.

[Bredereke, 1998] Bredereke, J. Requirements specification and design of a simplified telephone network by functional documentation. Technical Report 367, CRL, MacMaster University.

[Cerone, 2000] Cerone, A. Axiomatisation of an interval calculus for theorem proving. Technical Report 00-05, Software Verification Research Centre, University of Queensland.

[Fidge et al., 1998a] Fidge, C., Hayes, I., and Mahony, B. Defining differentiation and integration in Z. In Staples, J., Hinchey, M., and Liu, S., editors, *IEEE International Conference on Formal Engineering Methods (ICFEM '98)*, pages 64–73. IEEE Computer Society.

[Fidge et al., 1998b] Fidge, C., Hayes, I., Martin, A., and Wabenhorst, A. A set-theoretic model for real-time specification and reasoning. In Jeuring, J., editor, *Mathematics of Program Construction (MPC'98)*, volume 1422 of *LNCS*, pages 188–206. Springer-Verlag.

[Guerra, 1999] Guerra, S. *Defaults in the Specification of Reactive Systems*. PhD thesis, Department of Computer Science, University College London.

[Hayes, 1990] Hayes, I. Specifying physical limitations: A case study of an oscilloscope. Technical Report 167, Department of Computer Science, University of Queensland. Revised 1993.

[Hayes and Sanders, 1995] Hayes, I. and Sanders, J. Specification by interface separation. *Formal Aspects of Computing*, 7(4):430–439.

[Hayes and Utting, 1997] Hayes, I. and Utting, M. A sequential real-time refinement calculus. Technical Report 97-33, Software Verification Research Centre, University of Queensland.

[Heitmeyer, 1996] Heitmeyer, C. Requirements specifications for hybrid systems. In Alur, R., Henzinger, T., and Sontag, E., editors, *Hybrid Systems III*, volume 1066 of *LNCS*. Springer-Verlag.

[Mahony, 1992] Mahony, B. *The Specification and Refinement of Timed Processes*. PhD thesis, Department of Computer Science, University of Queensland.

[Mahony and Hayes, 1992] Mahony, B. and Hayes, I. A case-study in timed refinement: A mine pump. *IEEE Transactions on Software Engineering*, 18(9):817–826.

[Morgan, 1990] Morgan, C. *Programming from Specifications.* Prentice Hall.

[Problem Description, 2000] The Light Control Case Study: Problem Description. *Journal of Universal Computer Science*, Special Issue on Requirements Engineering (This Volume).

[Smith, 2000] Smith, G. Stepwise development from ideal specifications. In Edwards, J., editor, *Australasian Computer Science Conference (ACSC 00)*, volume 22(1) of *Australian Computer Science Communications*, pages 227–233. IEEE Computer Society.

[Swartout and Balzer, 1982] Swartout, W. and Balzer, R. On the inevitable intertwining of specification and implementation. *Communications of the ACM*, 25(7):438–440.

[van Lamsweerde et al., 1995] van Lamsweerde, A., Darimont, R., and Massonet, P. Goal-directed elaboration of requirements for a meeting scheduler: Problems and lessons learnt. In *Second IEEE Symposium on Requirements Engineering (RE'95)*, pages 194–203. IEEE Computer Society.

[van Schouwen et al., 1993] van Schouwen, A., Parnas, D., and Madey, J. Documentation of requirements for computer systems. In *First IEEE Symposium on Requirements Engineering (RE'93)*, pages 198–207. IEEE Computer Society.