Towards a Cookbook for Modelling and Refinement of Control Problems

Michael Butler

24 February 2009

1 Introduction

This document is a attempt to develop some guidelines on modelling control problems in Event-B. It is influenced in part by the Problem Frame approach of identifying phenomenon in the system. We identify two control problem patterns, the *autonomous* controller, which involves some plant and a controller, and the *commanded* controller, which includes an operator in addition. Within these patterns we identify several kinds of phenomenon and look at how they may be used to guide the construction of formal models and the refinement of formal models. The modelling concepts identifed are intended to encourage a focus on the problem rather than on the solution by identifying physical variables and operator commands at the abstract level. An important characteristic of controllers is the distinction between a physical phenomenon and the internal representation within a controller. We avoid making this distinction at the abstract level and introduce it through refinement patterns for sensors and actuators.

Here we only consider controllers from a discrete viewpoint. A continuous treatment is the subject of on-going investigations. It may be posible to incorporate results on the treatment of continuous behaviour later.

2 Autonomous controller

Let us assume a general system set up consisting of some plant (e.g., a pump in a steam boiler) and a controller for that pump. We assume there is no (human) operator involved so refer to this pattern as an *autonomous* controller. This is illustrated by the context diagram of Figure 1. In this case the plant represents the environment of the controller.

In the manner of Problem Frames, we would identify the phenomenon of the system such as a water level, or a pump status (*on*, *off*). Figure 1 illustrates that these phenomenon are shared between the controller and the plant (label *A*). Since the value/state of these phenomenon can change, we will model them as variables in our formal model. Since they represent physical phenomenon,

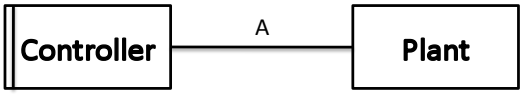

Figure 1: Autonomus Controller Pattern

we refer to them as *physical variables*. We can categorise physical variables as follows:

Uncontrolled physical variable: This type of phenomenon is not under the control of the controller and its state is determined by forces in the plant, e.g., the water level. We will use *up* to stand for uncontrolled physical variables.

Controlled physical variable: This type of phenomenon is under the control of the controller and its state is determined by the controller, e.g., the pump status. We will use *cp* to stand for controlled physical variables.

Note that by making the distinction in this way we are already biasing the problem description towards a controller viewpoint. This is not unreasonable since our job (presumably) is to build a controller that solves the given problem in the given environment.

Controlled and uncontrolled physical variables are incorporated into a B model simply as variables. The types of these variables should be determined by the nature of the phenomenon. For example, a water level would be typed as integer or real[1]. A pump status would be typed by an enumerated set that distinguishes the different status values.

Corresponding to the uncontrolled and controlled variables, we introduce two kind of events in the model, *plant events* and *control events*. Plant events can update the uncontrolled variables in a way that may depend on both the controlled and uncontrolled variables. Schematically they would be of the form:

$$\begin{array}{lcl} P & = & \textbf{when}\ \ G_P(up, cp) \\ & & \textbf{then}\ \ up := F_P(up, cp)\ \ \textbf{end} \end{array}$$

Note we use P for plant events and C for control events. For simplicity of presentation, the above scheme assumes a deterministic update. The scheme could easily be generalised to nondeterministic events.

It seems reasonable to assume a loose coupling between separate physical variables thus each plant event should update just one uncontrolled variable and we would have at least one plant event for each uncontrolled variable. It may be convenient to have more than one plant event for each physical variable in order to deal with different cases of update function. For example, in modelling the environmental update to the water level in a steam boiler, we may have separate cases for when the pump is on and when the pump is off.

[1]Real arithmetic is not currently support by Rodin.

Control events can update the controlled variables in a way that may depend on both the controlled and uncontrolled variables. Schematically they would be of the form:

$$\begin{array}{lcl} C & = & \textbf{when } G_C(up, cp) \\ & & \textbf{then } cp := F_C(up, cp) \textbf{ end} \end{array}$$

We could assume a tighter coupling between controlled variables than uncontrolled variables. Thus each control event could update several controlled variables (cp_1, cp_2, etc) simultaneously. As with plant events, we may have several cases for control events. For example, in a pump controller we might have one control event for when the water level is below a threshold and another for when it is above the threshold.

It may be convenient to partition the control events into separate responsibility groupings. An open question is whether responsibility groupings are based around (i) uncontrolled phenomenon or (ii) controlled phenomenon. An example of (i) would be having one group of control events responsible for responding to changes to the water level and another responsible for changes in tank pressure. An example of (ii) would be having one group of events responsible for managing the pump status and another for managing the alarm status.[2]

Sensing and actuating

In any control system, the controller gets (estimates of) the values of uncontrolled physical variables via sensors and the controller sets the values of controlled variables via actuators. At this level of abstraction, we have not explicitly modelled sensing and actuation events. These can be viewed as part of the *solution* to a control problem and thus we introduce them later via refinement (Sections 5 and 6).

Recap

In order to model an automomous controller we have identified the following Event-B modelling concepts:

- uncontrolled variables
- controlled variables
- plant events
- control events

[2] My current thinking is in favour of case (i) at an abstract level since it seems to be based more directly on the problem. Case (ii) feels more like a design solution.

3 Responsiveness

A critical consideration is how responsive we consider a controller to be to changes in the environment. Do we assume that the controller can respond to every (discrete) change in the uncontrolled variables? If we regard this assumption as being too strong, then we cannot specify very much about the controller response to changes in the uncontrolled physical variables (caused by environmental forces). This is precisely where the link between continuous and discrete behaviour influences the modelling of control problems. This remains an open question in the Event-B context, but we can make the assumption that the controller does indeed respond to every discrete change in the uncontrolled variables. We can enforce this responsiveness assumption in the model by introducing an auxiliary boolean variable up_r for each uncontrolled physical variable up. This is used to prevent an uncontrolled physical variable from being modified more than once in between control events. The schematic plant and controller events of Section 2 are extended as follows:

$$\begin{array}{lcl} P & = & \textbf{when}\ \ G_P(up, cp) \wedge up_r = F \\ & & \textbf{then}\ \ up := F_P(up, cp) \parallel up_r := T\ \ \textbf{end} \\ C & = & \textbf{when}\ \ G_C(up, cp) \\ & & \textbf{then}\ \ cp := F_C(up, cp) \parallel up_r := F\ \ \textbf{end} \end{array}$$

Note the asymmetry. This scheme allows many control events in between each plant event. This is weaker than requiring alternation between plant and controller but it is sufficient to encode the responsiveness assumption. It also further enforces our bias towards the controller in the problem description.

In the case where there is more than one uncontrolled physical variable, we would have one responsiveness auxiliary variable for each uncontrolled physical variable. To ensure that the controller is sufficiently responsive, we need to ensure that every plant change should lead to a controller response. This could be checked by having enabledness proof obligations on the control events to ensure that when a responsiveness variable up_r is true, then there is some control event that will set up_r to false, i.e., some control event that will respond to the change in up_r:

$$up_r = T \quad \Rightarrow \quad grd(C_1) \vee \cdots \vee grd(C_n)$$

Note that the encoding of responsiveness could be introduced as a (superposition) refinement of a more abstract model which does not incorporate responsiveness. Also it may be convenient to have several stage of augmentation (horizontal) refinement to introduce plant and controller features in a layer fashion before introducing the responsiveness assumption. These are open and subjective issues.

Recap

In order to model a responsiveness assumption in an automomous controller we have identified the following Event-B modelling concept:

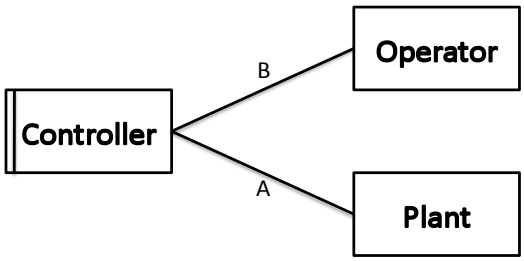

Figure 2: Commanded Controller Pattern

- auxiliary responsiveness flag

4 Commanded Controller

Now we extend the autonomous controller with a human operator as found, for example, in a mechanical press, a cruise control or a train control. This pattern is illustrated by the diagram of Figure 2. We refer to this as a *commanded controller* since the operator issues commands to the controller which influence the way it controls the plant. For example, in a cruise control, an operator (the driver) may switch the cruise control on or may modify the target speed.

To deal with the commanded controller, we identify two additional modelling concepts corresponding to phenomenon shared between the operator and the controller (B in Figure 2):

Command event: This is an event modelling a command by the operator. In a cruise control this would include commands such as *switch on* or *increase target speed*.

Commanded variable: This represents a variable whose value is determined by an operator command and that influences the way the controller controls the plant via controller events. For example, in a cruise control system we might have a *status* commanded variable with values $\{off, standby, active\}$ and a *target speed* commanded variable.

We use cm for commanded variable and M for command event. A command event may depend on several commanded variables and will update several commanded variables:

$$\begin{array}{lcl} M & = & \textbf{when } G_M(cm_1, cm_2) \\ & & \textbf{then } cm_1 := F_M(cm_1, cm_2) \textbf{ end} \end{array}$$

It is sometimes the case that a commanded variable is updated via several different commands, e.g, *increase* target speed, *decrease* target speed. In these cases it is sometimes useful to abstract these into a single command event that modifies the variable in some general or possibly nondeterministic way, e.g.,

abstract *increase* target speed and *decrease* target speed into a single *modify* target speed event. The distinctions between different cases of the same abstract command event can be introduced through refinement. The usefulness of doing this kind of command abstraction is that is leads to a smaller abstract model that 'distills' the essence of a control problem into a form that is easier to validate than having one which several events that are similar. This is a subjective point.

At this abstract level we are using the command events to model the way in which the controller *responds* to an operator command. We can deal with the controller *requesting* execution of a command separately via refinement (Section 7). In Section 3 we introduced the encoding of a responsiveness assumption because we separate the plant events that modify uncontrolled physical variables from the control events that modify controlled physical variables in response. Our treatment of command as events does not make a distinction between requesting a command and the controller response to that. This means we do not need to deal with responsiveness to commands at this level. This will be addressed in Section 7.

What is the rationale for focusing on the controller response to commands in an abstract model? It is based on the observation that this is where the distinctive properties of a control problem reside. The essence of the a particular control problem is describable in terms of physical variables (controlled and uncontrolled) and commanded variables. The identification of command events is determined by the identified commanded variables. The choice of command requests (as made by the operator) is in turn determined by the command events.

This is not to say that the separation of command requests and responses is uninteresting. Important issues to address include the delay between request and response, the possibility of errors (e.g., signal corruption) or combing several commands on to a single part of the interface (e.g., a single button serving several distinct functions in different commanded states). This issue does not seem to be specific to any particular control problem and the hope is that it can be dealt with in a generic way. Thus we can separate it into a later refinement.

Recap

In order to model a commanded controller we have identified the following Event-B modelling concepts which serve in addition to those for the autonomous controller:

- commanded variables
- command events

5 Introducing Sensing

As already stated, in any control system the controller reads the values of uncontrolled physical variables via sensors. This means we need to distinguish

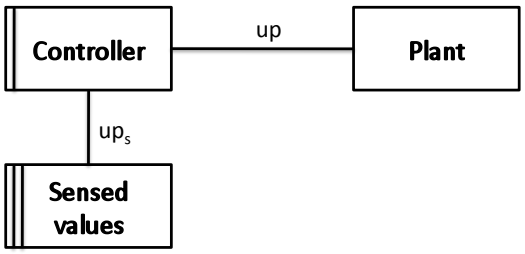

Figure 3: Pattern for introducing sensed values

between the value of an uncontrolled physical variable and the value of that variable as stored internally by the controller. We refer to the the internal controller version as a *sensed* variable. This is illustrated in the Problem Frame diagram of Figure 3 where the *sensed values* design domain is introduced. We view the introduction of the sensed variable as a design step and incorporate it via refinement. Along with a sensed variable, we also introduce a corresponding *sensor* event to set the sensed variable.

In Section 2 we saw the following general scheme for a plant event:

$$\begin{array}{lcl} P & = & \textbf{when } G_P(up, cp) \\ & & \textbf{then } up := F_P(up, cp) \textbf{ end} \end{array}$$

This plant event sets the value of uncontrolled variable up.

In a refinement step we want to introduce a sensed variable up_s corresponding to uncontrolled physical variable up. We also introduce a sensor event that copies the value of up to up_s. To model whether up and up_s correspond, we introduce an auxiliary flag up_{sf} that is set by sensor event for up. So the sensor event S corresponding to uncontrolled variable up is as follows:

$$S \quad = \quad up_s := up \;\; || \;\; up_{sf} := T$$

In Section 2 we saw the following general scheme for a control event:

$$\begin{array}{lcl} C & = & \textbf{when } G_C(up, cp) \\ & & \textbf{then } cp := F_C(up, cp) \textbf{ end} \end{array}$$

We refine a control event of this form so that it depends on the sensed variable up_s rather than uncontrolled physical variable up. We only allow the control event to fire if the sensed value is up to date (auxiliary flag up_{sf} is true) and we set this auxiliary flag back to false:

$$\begin{array}{lcl} C' & = & \textbf{when } G_C(up_s, cp) \;\wedge\; up_{sf} = T \\ & & \textbf{then } cp := F_C(up_s, cp) \;||\; up_{sf} := F \textbf{ end} \end{array}$$

We have that C' refines C and S refines $skip$. The correctness of the refinement relies on the following invariant:

$$Inv: \quad up_{sf} = T \quad \Rightarrow \quad up_s = up$$

To ensure this invariant is preserved, we need to assume that the environment does not change up in between the sensor event and the control event. This is encoded by strengthening the guard of the plant event so that up_{sf} is required to be false:

$$\begin{array}{lcl} P' & = & \textbf{when} \;\; G_P(up, cp) \;\wedge\; up_{sf} = F \\ & & \textbf{then} \;\; up := F_P(up, cp) \;\; \textbf{end} \end{array}$$

This refinement pattern for introducing sensed variables and sensor events is independent of the responsiveness pattern as described in Section 3 but it will also work together with the responsiveness pattern. Incorporating the responsiveness together with the sensing will result in plant events being followed by sensor events which will in turn be followed by control events.

The sensing pattern is applicable to both the autonomous and commanded controller patterns.

Recap

In order to introduce sensing in a model through refinement we have identified the following Event-B modelling concepts:

- sensed variables
- sensor events
- auxiliary sensor flag

6 Introducing Actuation

In the previous section we used refinement to separate the change to uncontrolled physical variables from the sensing of those physical variables. In this section we use refinement to introduce a distinction between a controller decision about controlled physical variables from the actuation of those physical variables. For example, in a steam boiler pump, a controller event might decide to switch the pump on. The controller will send a signal to the pump and a short while later the pump will respond to that signal by becoming active.

Now for every controlled physical variable cp we introduce a new *actuation* variable cp_a in which the value for cp decided by the controller is stored. The actuation variable is regarded as internal to the controller. This is illustrated in the Problem Frame diagram of Figure 4 where the *actuation values* design domain is introduced.

In Section 2 we saw the following general scheme for a control event:

$$\begin{array}{lcl} C & = & \textbf{when} \;\; G_C(up, cp) \\ & & \textbf{then} \;\; cp := F_C(up, cp) \;\; \textbf{end} \end{array}$$

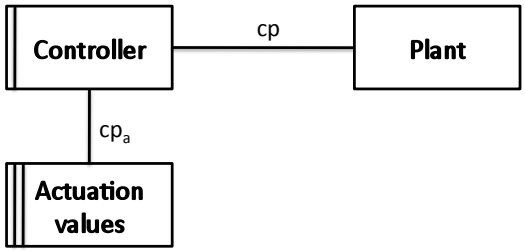

Figure 4: Pattern for introducing actuation values

In the refinement we introduce a control event C' that assigns to actuation variable cp_a instead of physical variable cp. We also introduce an auxiliary flag cp_{af} that is set to true when the actuation variable has been set by C':

$$\begin{array}{lcl} C' & = & \textbf{when } G_C(up, cp_a) \\ & & \textbf{then } cp_a := F_C(up, cp_a) \parallel cp_{af} := T \textbf{ end} \end{array}$$

We introduce an actuation event that copies the value of the actuation variable to the controlled physical variable:

$$\begin{array}{lcl} A & = & \textbf{when } cp_{af} = T \\ & & \textbf{then } cp := cp_a \parallel cp_{af} := F \textbf{ end} \end{array}$$

Now here is where the refinement set up differs from the sensing pattern. Instead of C' refining C it refines *skip* and the actuation event A refines the abstract control event C:

$$C' \textbf{ refines } skip, \qquad A \textbf{ refines } C$$

The reason for this arrangement is that we view an abstract control event as having a direct effect on a physical controlled variable[3]. With the actuation refinement pattern, the actuation of a controlled physical variable is achieved by the actuation event. The controller decision, as represented by the event C', has been internalised to the controller. The correctness of the refinement relies on the following invariants

$$\begin{array}{lrcl} Inv1: & cp_{af} = F & \Rightarrow & cp_s = cp \\ Inv2: & cp_{af} = T \ \wedge \ G_C(up, cp) & \Rightarrow & cp_a = F_C(up, cp) \end{array}$$

[3]An alternative would be to treat a controlled physical variable cp as a new variable introduced in the refinement and assume that the more abstract model refers to actuation variables rather than controlled physical variables. Then C' would refine C and the actuation events would refine *skip*. What I dislike about this alternative is that we would not be specifying controlled physical variables in the abstract model, only actuation variables which are internal to the controller. The abstract model should specify the problem not the solution.

Figure 5: Pattern for introducing operator buttons

To ensure this invariant is preserved, we need to assume that the environment does not change up in between the internal control event c' and the actuation event A. This is encoded by strengthening the guard of the plant event so that up_{af} is required to be false:

$$\begin{array}{rcl} P' & = & \textbf{when} \;\; G_P(up, cp) \;\wedge\; up_{sf} = F \\ & & \textbf{then} \;\; up := F_P(up, cp) \;\; \textbf{end} \end{array}$$

This actuation refinement pattern is independent of the responsiveness pattern as described in Section 3 but it will also work together with the responsiveness pattern. Incorporating the responsiveness together with the sensing and actuation pattens will result in plant events being followed by sensor events which will in turn be followed by internal control events which in turn will be followed by actuation events.

The actuation pattern is applicable to both the autonomous and commanded controller patterns.

Recap

In order to introduce actuation in a model through refinement we have identified the following Event-B modelling concepts:

- actuation variables
- internal controller events
- actuation events (refining abstract control events)
- auxiliary actuation flag

7 Introducing Operator Requests (Buttons)

In the commanded controller modelling pattern of Section 4 we identified the concepts of commanded variable and command event. We said that the command event models the response by the controller to a command from the operator. In this section we separate a command request by the operator from the response to that command by the controller. We assume that an operator requests commands by means of a button or some such device. This is illustrated by the diagram in Figure 5.

Recall from Section 4 that a command event has the form:

$$\begin{array}{lcl} M & = & \textbf{when}\ \ G_M(cm_1, cm_2) \\ & & \textbf{then}\ \ cm_1 := F_M(cm_1, cm_2)\ \ \textbf{end} \end{array}$$

Let us assume that the operator uses a button to request a command. Associated with a button b, we introduce an event B that is used to model the pressing of that button and a flag b_f that is set to true when the button b has been pressed and is set to false when the button press is responded to. The button press event has the following form:

$$\begin{array}{lcl} B & = & \textbf{when}\ \ b_f = F \\ & & \textbf{then}\ \ b_f := T\ \ \textbf{end} \end{array}$$

The link between the button b and a command event M is made by strengthening the guard of the command event so that auxiliary flag b_f is required to be true. The b_f flag is also set to false by the refined command event:

$$\begin{array}{lcl} M' & = & \textbf{when}\ \ b_f = T\ \wedge\ G_M(cm_1, cm_2) \\ & & \textbf{then}\ \ cm_1 := F_M(cm_1, cm_2)\ ||\ b_f := F\ \ \textbf{end} \end{array}$$

Our button modelling pattern incorporates a responsiveness assumption: once a button has been pressed, the controller will respond (modelled by the command events) before the button can be pressed again. If appropriatem this could be weakened to a model in which button presses are queued.

It some systems, the same button is used for several different functions. We can model this by using the same button flag b_f for different command events. For example we might have two command events Ma and Mb being linked with the same button b:

$$\begin{array}{lcl} Ma' & = & \textbf{when}\ \ b_f = T\ \wedge\ G_{Ma}(cm_1, cm_2) \\ & & \textbf{then}\ \ cm_1 := F_{Ma}(cm_1, cm_2)\ ||\ b_f := F\ \ \textbf{end} \\ Mb' & = & \textbf{when}\ \ b_f = T\ \wedge\ G_{Mb}(cm_1, cm_2) \\ & & \textbf{then}\ \ cm_1 := F_{Mb}(cm_1, cm_2)\ ||\ b_f := F\ \ \textbf{end} \end{array}$$

The case distinction between different usages of the same button will be made by the guards G_{Ma} and G_{Mb} respectively. If the guards overlap then the choice of command event will be nondeterministic between Ma and Mb. If the command events Ma and Mb have quite different effects, then having overlapping guards would be a poor operator interface design since the operator could not predict which effect pressing the button b will have.

The incorporation of the buttons could be treated as a synchronised composition between two Event-B machines, a buttons machine in parallel with the existing controller machine. In the buttons machine, the command event would simply depend on b_f:

$$\begin{array}{lcl} M & = & \textbf{when}\ \ b_f = T \\ & & \textbf{then}\ \ cm_1 := b_f := F\ \ \textbf{end} \end{array}$$

In the controller machine, the command event M would be as in the abstract model. In the composition of the buttons machine with the controller machine, the corresponding command events would be made to synchronise.

The treatment of the operator interface as boolean buttons could be enriched to incorporate values associated with operator commands (such as target speed value). This could be modelled, for example, by having a value variable b_v associated wth a button b. Other models for the interface could also be explored such as sets of signals from some rich datatype of signals.

We could incorporate error behaviour in the operator interface layer by introducing error events that set the button variables so some undesirable values.

8 Incorporating an operator display

An operator display is used to present values of commanded variables and possibly also physical variables to the operator. Treatment of this is yet to be added.

9 Decomposition

An important decomposition step is to separate a specification of the controller from the rest of the system (plant and operator). The sensing and actuation represent the interface between the controller and and the plant and operator. Introduction of the sensing and actuation mechanisms (via refinement) allows for the separation of the controller specifications from the overall (refined) model. Whether anything else is required before decomposing the controller from the rest of the system requires further exploration.