On Fitting A Formal Method Into Practice*

Rainer Gmehlich[1], Katrin Grau[1], Stefan Hallerstede[2], Michael Leuschel[2], Felix Lösch[1], and Daniel Plagge[2]

[1] Robert Bosch GmbH, Stuttgart, Germany
[2] Heinrich-Heine-University of Düsseldorf, Germany

Abstract. The development of the Event-B formal method and the supporting tools Rodin and ProB was guided by practical experiences with the B-Method, the Z specification notation, VDM and similar practical formal methods. The case study discussed in this article — a cruise control system — is a serious test of industrial use. We report on where Event-B and its tools have succeeded, where they have not. We also report on advances that were inspired by the case study. Interestingly, the case study was not a pure formal methods problem. In addition to Event-B, it used Problem Frames for capturing requirements. The interaction between the two proved to be crucial for the success of the case study. The heart of the problem was tracing informal requirements from Problem Frames descriptions to formal Event-B models. To a large degree, this issue dictated the approach that had to be used for formal modelling. A dedicated record theory and dedicated tool support were required. The size of the formal models rather than complex individual formulas was the main challenge for tool support.

1 Introduction

This article recounts an attempt to apply Event-B [1] to an industrial specification problem in a methodologically heterogenous environment without prior use of formal methods. This required integration within an existing, evolving development methodology. This means the methodology cannot be dictated to follow customs and conventions that have arisen within formal methods such as Event-B.

We believe some of the problems we encountered will be typical for industrial deployment of formal methods in general. In particular, this concerns problems related to the different cultures, customs and conventions in industry and academia. At times, these problems (appear to) become more severe than all technical problems taken together. It seems advisable to be prepared for this, assuming the different cultures are unavoidable.

* This research was carried out as part of the EU FP7-ICT research project DEPLOY (Industrial deployment of advanced system engineering methods for high dependability and productivity) http://www.deploy-project.eu.

The article presents a non-chronological digest of an experiment carried out at Bosch: to develop a model of a cruise control system. The focus of this experiment was to apply Event-B to an industrial problem in an industrial environment. In that environment, Problem Frames [8] are used to deal with informal requirements. Initially, the academics did not yet grasp the central importance of this: formal methods seem to have a tendency to take over whole development processes and dictate what should be done instead. However, here formal methods were only a piece in a bigger puzzle. Other parts of the puzzle dealt with continuous control (which is provided by specialised design methods) or real time (which is considered more a matter of code generation). Note that such issues enter the case study in the form of assumptions at various places. We have learned that excluding parts of a problem can be a challenge for formal methods. The claim of correctness will only be with respect to a large set of assumptions, the justifications of which are unavailable.

The core of the formal modelling problem of the experiment concerns only the switching behaviour of the cruise control. The main difficulties are to fit Event-B into the overall development scheme, while finding the appropriate abstractions to express the formal model of the cruise control. In [16], a formal development of a cruise control in Event-B is presented. The development is more ambitious than our development, but it does neither support traceability of Problem Frames descriptions nor hazard analysis. It also does not respect the boundaries described above about what not to model. One would also think that hybrid modelling of the cruise control (for instance, [15]) should be of advantage. But again, our problem is the switching behaviour of the cruise control. We want to avoid duplicating work already carried out by other engineers. In principle, one could also use an approach based on StateMate [**?**]. We believe, however, that tracing of requirements and matching formal models with Problem Frames would be difficult. In particular, although Event-B refinement is not satisfactory for matching Problem Frames elaboration, it gave us a means to express what we wanted.

We believe our observations and findings would apply in similar form to other formal methods used in place of Event-B. In this context, note again the exclusion of certain aspects such as continuity or real time. Not everything is under control of the formal method. Our contribution is an approach for tracing requirements by matching Problem Frames with formal Event-B models using a dedicated way of modelling and instantiating records. We also have made progress in checking large models for deadlocks using constraint-solving techniques.

Industrial use is impossible without having supporting tools. For Event-B the Rodin tool [2] has been developed. In short, it deals with editing and proof obligation generation. It has a plugin architecture that permits its extension with new functionality. One plugin provided is ProB [11], a tool for model animation, model checking and constraint checking.

Overview. Section 2 describes how Event-B was fitted into an industrial development process. The following sections focus on specific problems of the integration. Section 3 discusses integration with Problem Frames. Section 4 dis-

cusses a specific aspect of that integration: the relationship of Problem Frames elaboration and Event-B refinement. Section 5 discusses a scaling problem by way of deadlock analysis. A conclusion follows in Section 6.

2 Event-B in an Industrial Development Process

The case study we discuss in this article is carried out within the Deploy research project. The project's main objective is to make major advances in engineering methods for dependable systems through the deployment of formal engineering methods. One work package of the project deals with the deployment of formal methods, in particular, Event-B and ProB in the automotive sector. The case study applies Event-B to the modelling of a *cruise control system*. A cruise control system is an automotive system implemented in software which automatically controls the speed of a car. It is part of the engine control system which controls actuators of the engine (such as injectors, fuel pumps or throttle valve) based on the values of specific sensors (such as the accelerator pedal position sensor, the airflow sensor or the lambda sensor).

The cruise control system consists of a discrete part describing the control logic and continuous parts describing the actual closed loop controllers required to adjust the speed of the car. In the case study we focus exclusively on the discrete part, that is, the switching behaviour of the cruise control system. The continuous control part is provided by other design methods.

2.1 Event-B

Event-B models are composed of *contexts* and *machines*. Contexts capture static aspects of a model expressed in terms of *carrier sets*, *constants* and *axioms*. Consequences of the axioms can be stated as *theorems* (that need to be proved). Fig. 5 and 7 below show two contexts. The *concrete* context of Fig. 7 is said to *extend* the *abstract* context of Fig. 5: all carrier sets, constants, axioms and theorems of the context being extended are visible in the extending context.

Machines capture dynamic aspects of a model (see Fig. 6). The state of a machine is described in terms of *variables*. The possible values of the variables are constrained by *invariants* (see *inv1* and *inv2* of Fig. 6). Possible state changes are modelled by *events*. Each event consists of a collection of *parameters* p, of *guards* g and of *actions* a (a collection of simultaneous update statements). We use the following schema to describe events: any p when g then a end. An event may cause a state change if its guard is true for a choice of parameters. Event-B does not make any fairness assumptions about event occurrences. *Refinement* is used to specify more details about a machine. For instance, the *concrete* machine of Fig. 8 is a refinement of the *abstract* machine of Fig. 6. The state of the abstract machine is related to the state of the concrete machine by a *gluing invariant* associated with the concrete machine that relates abstract variables to concrete variables (see invariant *inv3* in Fig. 8). Each event of the abstract machine is *refined* by one or more concrete events. Roughly speaking, the events of the

abstract machine must be capable of simulating the behaviour of the events of the concrete machine. The Rodin tool can generate proof obligations to verify properties such as invariant preservation or refinement.

2.2 Fitting Event-B into Development Practice

Introducing the formal development method Event-B into industrial practice (in the automotive sector) requires integration with existing development processes and tools. Fig. 1 shows a sketch of a development process which includes Event-B. This article deals with the use of Event-B at the position indicated in the figure. The main challenge encountered in this respect is concerned with the relationship of Event-B and Problem Frames.

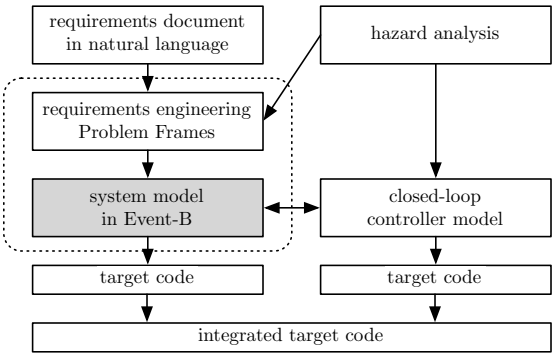


Fig. 1. Overview of the development process

We comment briefly on the phases of the development process, to give an impression of the general picture and the constraints imposed. Development begins with natural language requirements that tend to be unstructured and difficult to relate to a formal model. A hazard analysis yields safety requirements to be incorporated. Requirements engineering and hazard analysis go hand in hand. The development of the closed loop controller is done by control engineers. Verifying closed loop controllers requires reasoning about continuous time behaviour. Since specialised methods are already in place we did not study modelling of continuous time behavior in the case study. We modelled the discrete part of the system in Event-B making a contribution to the existing design process. Another work package of the Deploy project focused on the task of generating code from Event-B models [3].

The need for Problem Frames for requirements engineering had been recognised early on with formal system modelling in view. The decision to use Problem Frames was made after a first attempt to directly model natural language requirements in Event-B resulted in a large gap between the requirements document and the Event-B model. The gap between the two documents was too large to permit maintaining them consistently: it is indispensable to validate by means of a review, say, that an Event-B model adequately captures what is stated in the requirements. And tracing this information seemed out of reach. Furthermore, the Bosch engineers had made good experiences with using the Problem Frames approach for structuring requirements as well. It is important to note here, that the conditions determining a decision to use a certain method

in industry are considerably different from those in academia: the method chosen in industry must not only fit a single problem it must also be understandable for a large variety of engineers who are not directly involved in solving the problem at hand. The Problem Frames approach looked very promising to the Bosch engineers because it could easily be understood by the development engineers.

The Problem Frames approach enabled the Bosch engineers to validate the Event-B model with respect to the requirements and provided an easier way of tracing requirements in the Event-B model. With the introduction of the Problem Frames approach they obtained two simpler validation problems:

(1) to validate whether the problem frames capture the natural language requirements

(2) to validate whether the Event-B model corresponds to the problem frames.

Each of the two validation problems appeared to be feasible as opposed to the direct approach from natural language requirements to Event-B models.

The insight we gained from the early phase of the case study is that introducing Event-B in industry on its own is difficult. Introducing Event-B in conjunction with supporting Problem Frames greatly reduces the entry barrier for engineers to use Event-B. Similar observations have been made with UML-B before [14].

2.3 Problem Frames

Problem Frames is an informal graphical requirements engineering method developed by Michael Jackson [8]. The immediate focus of Problem Frames, as its name suggests, is on software development as a problem to be solved. The problem to be solved is hereby visualized using *problem diagrams* that contain a *machine*, i.e., the system to be built, the *problem world*, i.e., the environment the system is interacting with and the *requirements* which are expressed in terms of the problem world. The requirements engineering process usually starts with a *context diagram*, an abstract problem diagram, which describes the main elements of the problem world as well as the overall requirement the system shall fulfill. Fig. 2 shows a simple *context diagram* of the cruise control system.

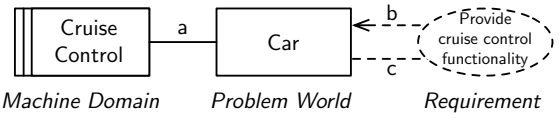


Fig. 2. Context diagram for cruise control system.

The machine interacts with the problem world by *shared phenomena* at the interface *a*. Typically, these *phenomena* are events and states, controlled either by the problem world or by the machine and shared at input-output ports of the machine. Requirements are described only in terms of the phenomena of the problem world which are shared between the problem world and the requirements at interfaces *b* and *c*.

Projections of this context diagram, called *subproblems*, are then used to describe different aspects of the problem. A more detailed description of the concepts of Problem Frames can be found in [8].

We extended the original Problem Frames approach [9,13] by an additional operation called *elaboration* as well as an adapted version of the already existing projection operation. In an *elaboration* of an abstract problem diagram the environment, the phenomena and the requirements are described in more detail. For example, the problem diagram in Fig. 4 is an elaboration of the problem diagram in Fig. 3. Elaboration in Problem Frames thus serves a similar purpose as refinement in Event-B, i.e., to relate abstract descriptions of the system to more concrete descriptions.

2.4 The Cruise Control System

In the following we describe the control logic of the cruise control system in more detail. The behaviour of the cruise control system is determined by three different operating modes: NOCTRL, CTRL, ACTRL. In the NOCTRL mode the system is inactive, that is, it is not actively regulating the speed of the car. In the CTRL mode the system is either maintaining or approaching a previously defined target speed. In the ACTRL mode the system is either accelerating or decelerating by a predefined value. The three modes of the system can be switched by the driver using the *control interface* or by the software in case the control software detects an error. In the latter case the mode is always switched to NOCTRL.

There are two ways of a driver to control the behaviour of the cruise control system: (i) using the brake pedal or clutch pedal to (temporarily) deactivate the cruise control system, and (ii) using the control elements provided by the operating lever. The operating lever usually has the following buttons: (a) SET to define a target speed, (b) RESUME to resume a previously defined target speed, (c) TIPUP to increase the target speed, (d) TIPDN to decrease the target speed, (e) ACC to accelerate, (f) DEC to decelerate. Furthermore, there is a dedicated switch for switching the cruise control system ON or OFF.

Depending on commands given by the driver or signals received by sensors the cruise control system switches between the modes. In order to distinguish the different operational states of the system the three major modes are further partitioned into a number of ten submodes as shown in Table 1.

3 Relating Problem Frames to Event-B Models

A major obstacle during the case study was to understand how the gap between Problem Frames and Event-B could be closed. If the informal requirements could not be traced into the formal models, the development method would be of no use for the engineers. A close correspondence between concepts of Problem Frames and Event-B was needed to arrive at a systematic approach to requirements tracing. Feedback from the analysis of the formal models should suggest

Mode	Submode	Description
NOCTRL	UBAT_OFF	Ignition is off and engine not running
	INIT	Ignition is on and cruise control is being intialized
	OFF	Ignition is on, cruise control initialized and switched off
	ERROR	An irreversible error has occurred
	STANDBY	Cruise control has been switched on
	R_ERROR	A reversible error has occurred
CTRL	CRUISE	Cruise control is maintaining the target speed
	RESUME	Target speed is approached from above or from below
ACTRL	ACC	Cruise control is accelerating the car
	DEC	Cruise control is decelerating the car

Table 1. Modes and submodes of the cruise control system.

improvements to the informal requirements. For example, missing requirements were identified by using deadlock checking in ProB.

The central concern was to relate the elaboration and projection operations provided by the extended Problem Frames approach to the notion of refinement in Event-B. In order to illustrate this problem we use a small example which describes a fragment of the cruise control system both at an abstract level (see Fig. 3) and a more concrete (elaborated) level (see Fig. 4). Note that some aspects (e.g. Ignition) are ignored.

3.1 Problem Frames Description of the Cruise Control

Fig. 3 shows an abstract problem diagram of the pedal subproblem for the cruise control system. The diagram of Fig. 3 shows the machine domain CrCtl Pedals, the given domain Pedals, the designed domain State Model, and the requirement R1. The phenomena shared between the machine, the domains, and the requirement are shown in Fig. 3 under *interfaces* with the following syntax: *[Name of the interface]: [Domain controlling the phenomenon]!{list of shared phenomena}*. For example, the line "L1 : Pedals!{P_Env_PedalSignals}" means that the phenomenon *P_Env_PedalSignals* controlled by the domain Pedals is shared with the machine CrCtl Pedals. The requirement R1 states that "depending on the status of the pedals (P_Env_PedalSignals), the internal mode of the cruise control system (P_CrCtl_Mode) should change to the mode R_ERROR or ERROR."

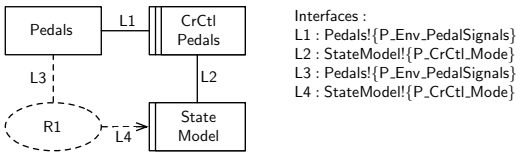
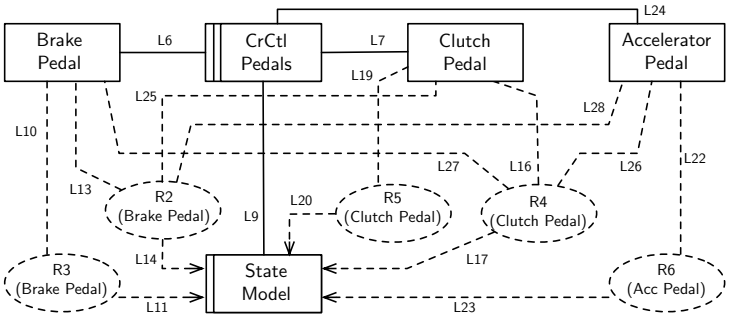


Fig. 3. Abstract problem diagram for pedals.

Fig. 4 shows the elaborated version of the abstract problem diagram of Fig. 3. The abstract given domain Pedals has now been elaborated into the



Interfaces :
L6 : BrakePedal!{P_Env_PS_BRK_PRSD, P_Env_PS_BRK_ERR}
L7 : ClutchPedal!{P_Env_PS_CLT_PRSD, P_Env_PS_CLT_ERR}
L9 : StateModel!{P_CrCtl_Mode}
L10 : BrakePedal!{P_Env_PS_BRK_ERR}
L11 : StateModel!{P_CrCtl_Mode}
L13 : BrakePedal!{P_Env_PS_BRK_PRSD, P_Env_PS_BRK_ERR}
L14 : StateModel!{P_CrCtl_Mode}
L16 : ClutchPedal!{P_Env_PS_CLT_PRSD, P_Env_PS_CLT_ERR}
L17 : StateModel!{P_CrCtl_Mode}
L19 : ClutchPedal!{P_Env_PS_CLT_ERR}
L20 : StateModel!{P_CrCtl_Mode}
L22 : AcceleratorPedal!{P_Env_PS_ACC_ERR}
L23 : StateModel!{P_CrCtl_Mode}
L24 : AcceleratorPedal!{P_Env_PS_ACC_ERR}
L25 : ClutchPedal!{P_Env_PS_CLT_ERR}
L26 : AcceleratorPedal!{P_Env_PS_ACC_ERR}
L27 : BrakePedal!{P_Env_PS_BRK_ERR}
L28 : AcceleratorPedal!{P_Env_PS_ACC_ERR}

Fig. 4. Elaborated problem diagram for pedals.

given domains Brake Pedal, Clutch Pedal, and Accelerator Pedal as well as the abstract phenomena shown in Fig. 3. For example, the abstract phenomenon P_Env_PedalSignals has been elaborated into the concrete phenomena P_Env_BRK_PRSD, P_Env_BRK_ERR, P_Env_CLT_PRSD, P_Env_CLT_ERR, and P_Env_PS_ACC_ERR. The requirement R1 of Fig. 3 has also been elaborated into the requirements R2, R3, R4, R5, R6.

For illustration we state the requirements R2 and R3. Requirement R2 is: "If the brake pedal is pressed (P_Env_PS_BRK_PRSD) and no pedal error is present (P_Env_PS_BRK_ERR = FALSE, P_Env_PS_CLT_ERR = FALSE, P_Env_PS_ACC_ERR = FALSE), the mode (P_CrCtl_Mode) must change to R_ERROR." Requirement R3 is: "If a brake pedal error is present (P_Env_PS_BRK_ERR), the mode (P_CrCtl_Mode) must change to ERROR." R4 states the same as R2 for the clutch pedal (P_Env_CLT_PRSD) and R5 the same as R3 for the clutch pedal error (P_Env_PS_CLT_ERR). R6 states the same as R3 for the accelerator pedal error (P_Env_PS_ACC_ERR). Pressing the accelerator pedal (P_Env_PS_ACC_PRSD) is part of a different subproblem and therefore not dealt with in Fig. 4.

3.2 Relating Problem Frames Concepts to Event-B Concepts

To support traceability of requirements in problem diagrams to formal elements of Event-B models we relate Problem Frames concepts one-to-one to Event-

B concepts. This simple approach to relating Problem Frames descriptions to Event-B models is a key to the feasibility problem of requirements tracing mentioned in Section 2.2. Table 2 indicates how the concepts are matched. The tran-

Problem Frames	**Event-B**
Problem diagram	Machine and context
Phenomena	Variables, constants or carrier sets
Types of phenomena	Carrier sets or constants
Requirements	Events and/or invariants
Elaboration of a problem diagram	Refinement of a machine or context
Projection of a problem diagram	Decomposition of a machine or context
Elaboration of phenomena	Data refinement

Table 2. Mapping of Problem Frame Elements to Event-B Elements

sition from Problem Frames descriptions to Event-B models is still a manual step in the design process. However, starting from the natural language requirements this intermediate step provides enough guidance to obtain a suitable Event-B model that captures the requirements and permits tracing them into that model.

The first three correspondences for problem diagrams, phenomena and their types do not pose problems. Requirements state properties that must hold for the system: if they are dynamic, they are best matched by events; if they are static, they are best matched by invariants. Determining the relationship of elaboration and refinement is more involved. In particular, an approach to record instantiation was needed on top of Event-B refinement for the correspondence to work. Dealing with projection and decomposition is still more complicated because in Problem Frames projection is used as early as possible in system description whereas in Event-B decomposition is usually delayed so that system-wide properties can be verified before decomposition.

3.3 A Matching Event-B Model of the Cruise Control

Using the mapping from Problem Frames to Event-B described in Section 3.2 we developed an Event-B model of the cruise control. Problem Frames are used to structure requirements and support their (informal) analysis. Requirements and domain properties can be stated in any way that appears convenient like Table 1. Fig. 5 shows a context that captures the table with OK representing all submodes except R_ERROR and ERROR.[1] The two constants PS_SET and PS_ERROR are abstractions for specific pedal signal combinations not expressible in the abstract model. The abstract machine (that corresponds to the abstract diagram of Fig. 3) is shown in Fig. 6. Depending on the pedal signals

[1] We have abstracted from the remaining submodes for the sake of brevity.

[2] The predicate *partition* states that the sets OK, {R_ERROR} and {ERROR} are a set-theoretical partition of the set T_Mode.

constants OK R_ERROR ERROR PS_SET PS_ERROR
sets T_Mode T_Env_PedalSignals
axioms @*axm1* PS_SET $\subseteq$ T_Env_PedalSignals
@*axm2* PS_ERROR $\subseteq$ T_Env_PedalSignals
@*axm3* PS_SET $\cap$ PS_ERROR $= \varnothing$
@*axm4* *partition*(T_Mode, OK, {R_ERROR}, {ERROR}) [2]

Fig. 5. Context of abstract model

variables P_Env_PedalSignals P_CrCtl_Mode
invariants @*inv1* P_Env_PedalSignals $\in$ T_Env_PedalSignals
@*inv2* P_CrCtl_Mode $\in$ T_Mode
events
event CrCtl_Chg_Mode_PedalSignals_R1
when P_Env_PedalSignals $\in$ PS_SET $\vee$ P_Env_PedalSignals $\in$ PS_ERROR
then P_CrCtl_Mode $:\in$ {R_ERROR, ERROR} end

Fig. 6. Abstract machine

the mode is changed. This is stated informally in the Problem Frame diagram of Fig. 3. Close correspondence between the diagram and the Event-B model is important for traceability of the requirements and validation of the formal model.

Correspondence of the abstract diagrams and models is straightforward. Relating elaborated problem diagrams to Event-B models is less obvious because close correspondence remains crucial. Refinement permits to introduce more details into a model as needed for elaboration. However, it does not allow to relate abstract and concrete phenomena systematically. What is needed is closer to "instantiation" of abstract phenomena by concrete phenomena. Refinement is powerful enough to emulate the intended effect of such an "instantiation": we can state the mathematics of it (see Fig. 7). Although refinement does not suit

constants iEnv_PS iEnv_PS_SET iEnv_PS_ERROR
axioms
@*axm5* iEnv_PS $\in \mathbb{B} \times \mathbb{B} \times \mathbb{B} \times \mathbb{B} \times \mathbb{B} \rightarrowtail$ T_Env_PedalSignals
@*axm6* iEnv_PS_ERROR $=$ iEnv_PS^{-1}[PS_ERROR]
@*axm7* iEnv_PS_ERROR $=$
$(\mathbb{B} \times \{\mathbf{T}\} \times \mathbb{B} \times \mathbb{B} \times \mathbb{B}) \cup (\mathbb{B} \times \mathbb{B} \times \mathbb{B} \times \{\mathbf{T}\} \times \mathbb{B}) \cup (\mathbb{B} \times \mathbb{B} \times \mathbb{B} \times \mathbb{B} \times \{\mathbf{T}\})$
theorem @*thm1* $\forall a, b, c, d \cdot$
iEnv_PS$(a \mapsto b \mapsto c \mapsto d \mapsto e) \in$ PS_ERROR $\Leftrightarrow b = \mathbf{T} \vee d = \mathbf{T} \vee e = \mathbf{T}$
@*axm8* iEnv_PS_SET $=$ iEnv_PS^{-1}[PS_SET]
@*axm9* iEnv_PS_SET $= (\{\mathbf{T}\} \times \{\mathbf{F}\} \times \mathbb{B} \times \{\mathbf{F}\} \times \{\mathbf{F}\}) \cup (\mathbb{B} \times \{\mathbf{F}\} \times \{\mathbf{T}\} \times \{\mathbf{F}\} \times \{\mathbf{F}\})$
theorem @*thm2* $\forall a, b, c, d, e \cdot$ iEnv_PS$(a \mapsto b \mapsto c \mapsto d \mapsto e) \in$ PS_SET $\Leftrightarrow$
$(a = \mathbf{T} \wedge b = \mathbf{F} \wedge d = \mathbf{F} \wedge e = \mathbf{F}) \vee (b = \mathbf{F} \wedge c = \mathbf{T} \wedge d = \mathbf{F} \wedge e = \mathbf{F})$

Fig. 7. Context of concrete model

our needs perfectly we can use it to develop the notion of "elaboration in Event-B" that will suit it. Here refinement is used for the development of a development method. Refinement in its generality may not be part of the final development method that could be used at Bosch for the modelling of discrete systems. The general technique refinement allowed us experiment with different notions using an available software tool, the Rodin tool. We are aware that the approach using refinement for instantiation that we describe in the following would be too complicated to scale. Of course, we intend that records and record instantiation be incorporated directly into the formal notation of Event-B so that refinement would not have to be used to imitate it. However, we found it instructive to determine first what kind of record concept would be needed for our purposes.

Our main insight here is that refinement seems to function as an enabler for the invention of the required technology. It may not itself be the required technology. The development of the concept of elaboration in Event-B is discussed in the next section.

variables P_Env_PS_BRK_PRSD P_Env_PS_BRK_ERR P_Env_PS_CLT_PRSD
P_Env_PS_CLT_ERR P_Env_PS_ACC_ERR P_CrCtl_Mode

invariants @*inv3* P_Env_PedalSignals = iEnv_PS(
P_Env_PS_BRK_PRSD $\mapsto$ P_Env_PS_BRK_ERR $\mapsto$ P_Env_PS_CLT_PRSD $\mapsto$
P_Env_PS_CLT_ERR $\mapsto$ P_Env_PS_ACC_ERR)

events

event CrCtl_Chg_Mode_PedalSignals_R2 refines CrCtl_Chg_Mode_PedalSignals_R1
when P_Env_PS_BRK_PRSD = **T** $\wedge$ P_Env_PS_BRK_ERR = **F** $\wedge$
P_Env_PS_CLT_ERR = **F** $\wedge$ P_Env_PS_ACC_ERR = **F**
then P_CrCtl_Mode := R_ERROR end

event CrCtl_Chg_Mode_PedalSignals_R3 refines CrCtl_Chg_Mode_PedalSignals_R1
when P_Env_PS_BRK_ERR = **T** then P_CrCtl_Mode := ERROR end

event CrCtl_Chg_Mode_PedalSignals_R4 refines CrCtl_Chg_Mode_PedalSignals_R1
when P_Env_PS_CLT_PRSD = **T** $\wedge$ P_Env_PS_BRK_ERR = **F**
P_Env_PS_CLT_ERR = **F** $\wedge$ P_Env_PS_ACC_ERR = **F**
then P_CrCtl_Mode := R_ERROR end

event CrCtl_Chg_Mode_PedalSignals_R5 refines CrCtl_Chg_Mode_PedalSignals_R1
when P_Env_PS_CLT_ERR = **T** then P_CrCtl_Mode := ERROR end

event CrCtl_Chg_Mode_PedalSignals_R6 refines CrCtl_Chg_Mode_PedalSignals_R1
when P_Env_PS_ACC_ERR = **T** then P_CrCtl_Mode := ERROR end

Fig. 8. Concrete machine

4 Elaboration in Event-B

Dealing with elaboration in Event-B was the central problem to be solved to deal adequately with traceability of requirements from problem diagrams. Its

solution required a dedicated record theory, an instantiation method and a tool improvement that could master the refinement-based modelling of the former.

A convention for modelling records. When elaborating Problem Frame diagrams each abstract phenomenon is replaced by a set of more concrete phenomena. For example, in the cruise control model the abstract phenomenon P_Env_PedalSignals is replaced by the concrete phenomena

P_Env_PS_BRK_PRSD	Brake pedal pressed,
P_Env_PS_BRK_ERR	Brake pedal error,
P_Env_PS_CLT_PRSD	Clutch pedal pressed,
P_Env_PS_CLT_ERR	Clutch pedal error,
P_Env_PS_ACC_ERR	Accelerator pedal error.

This problem appeared to be solved in the form of existing record theories for Event-B [4] or VDM [10]. However, after some experimentation with [4] we also shied away from trying [10]. The theories appeared too powerful. Simple facts —and we only needed simple facts— were comparatively difficult to prove. We wanted them to be proved automatically without further interaction. It took a while until we realised that we should formulate a simple effective theory of limited expressiveness that would satisfy our needs (but not more). The approach that we use now (see Fig. 7 and 8) does permit nearly fully automatic proofs. We use a restricted form of data refinement for records based on a convention to model records loosely by lists of variables. Refinement of such records is done by instantiating abstract variables by lists of concrete variables and similarly for parameters. The instantiation is expressed by means of invariants, for instance, *inv3* in Fig. 8. Function iEnv_PS is a bijection from the concrete variables to the abstract variables.[3] This function facilitates all refinement proofs. By means of it abstract constants are mapped to concrete constants that specify relationships between data values in more detail. For instance, by *axm6* of Fig. 7 the concrete set iEnv_PS_ERROR corresponds to the abstract set PS_ERROR. The concrete set iEnv_PS_ERROR is then specified in more detail in *axm7*. Theorem *thm1* formulates the set-theoretical equation in terms of an equivalence. In fact, theorem *thm1* is more useful in proofs. The approach of specifying iEnv_PS_ERROR by two axioms has been chosen in order to avoid introducing contradicting statements in contexts. Important facts such as theorem *thm1* are proved. Note that the shape of the theorems matches the needs of the refinement proof. For instance, theorem *thm2* is geared towards the refinement proof of event CrCtl_Chg_Mode_PedalSignals_R2. Letting a = P_Env_PS_BRK_PRSD, b = P_Env_PS_BRK_ERR and so on, using *inv3* we can infer

$$\begin{array}{l} \quad \text{P\_Env\_PS\_BRK\_PRSD} = \mathbf{T} \wedge \text{P\_Env\_PS\_BRK\_ERR} = \mathbf{F} \wedge \\ \quad \text{P\_Env\_PS\_CLT\_PRSD} = \mathbf{T} \wedge \text{P\_Env\_PS\_CLT\_ERR} = \mathbf{F} \wedge \\ \quad \text{P\_Env\_PS\_ACC\_ERR} = \mathbf{F} \\ \Rightarrow \text{P\_Env\_PedalSignals} \in PS\_SET \ , \end{array}$$

which establishes that the concrete guard implies the abstract guard. We have solved the problem concerning refinement *proofs* of record instantiation.

[3] This is even stronger than functional data-refinement.

Tool issues. The full model of the cruise control turned out to be difficult to model check. The extra theory provided in the contexts for the instantiation reduced the efficiency of the associated tools used for model checking and constraint solving (see Section 5). To solve the problem of instantiation a preprocessing step is applied to the model to automatically detect records usage. We begin by searching for axioms of the form $iAP \in CP \rightarrowtail\!\!\!\!\!\rightarrow AP$ where iAP is a constant, the "instantiation mapping", AP a carrier set, the abstract phenomenon to be instantiated, and CP a Cartesian product, modelling the list of concrete phenomena. Axiom @*axm5* in the concrete model above (Fig. 7) is such an axiom. Using $iAP \in CP \rightarrowtail\!\!\!\!\!\rightarrow AP$ we can safely assume the set AP to be equal to CP because of the existence of the bijection iAP. Technically, the concerned axiom is removed, the set AP turned into a constant and the axioms $AP = CP$ and $iAP = \mathrm{id}(CP)$ are added.

A method for records and record instantiation. The mathematical model and its treatment by the tool can be used to formulate a method for dealing with records and instantiation: All records are non-recursive. They may contain some constraints, for instance, "maximal speed" > "minimal speed". For the instantiation of fields, we simply state which concrete fields instantiate which abstract fields. Constants like PS_ERROR can be instantiated by specifying a corresponding subset of the concrete record. Abstract properties like @*axm3* have to be proved for the instantiated subsets.

Discussion. We do not believe that a general theory and method can be found that would satisfy all the needs of different industrial domains. We need specific theories and tools that work well in specific domains, for instance, the automotive domain. Similar experiences have been made in the railway domain at Siemens Transportation Systems [12], where ProB was improved to deal with large relations and sets. These arose in the modelling and validation of track topologies. Our quiet hope is that still some theory and technology can be shared. It just does not seem reasonable anymore to seek expressly a general theory with supporting technology.

5 Verifying Deadlock Freedom

Besides invariant preservation, the absence of deadlocks is crucial in this case study, as it means that the engineers have thought of every possible scenario. In other words, a deadlock means that the system can be in a state for which no action was foreseen by the engineers.

Deadlock freedom. An event is *enabled* in a state if there are values for its parameters p that make its guard g true in that state. We denote the *enabling predicate* $(\exists p \cdot g)$ of an event e by G_e. Event-B provides a way to verify the deadlock freedom of model: the (DLF) proof obligation of [1]: $A \wedge I \Rightarrow G_{e_1} \vee \ldots \vee G_{e_n}$, where A are the axioms, I are invariants and G_{e_ℓ} $(\ell \in 1 \,..\, n)$ the enabling predicates of the events e_ℓ of the considered machine. For the machine

of Fig. 8 this proof obligations is:[4]

$$
\begin{array}{l}
\text{“all axioms and theorems of Fig. 5 and Fig. 7”} \wedge \\
\text{“all invariants of Fig. 6 and Fig. 8”} \\
\Rightarrow (\mathrm{P\_Env\_PS\_BRK\_PRSD} = \mathbf{T} \wedge \mathrm{P\_Env\_PS\_BRK\_ERR} = \mathbf{F} \wedge \\
\quad \mathrm{P\_Env\_PS\_CLT\_ERR} = \mathbf{F} \wedge \mathrm{P\_Env\_PS\_ACC\_ERR} = \mathbf{F}) \vee \\
\quad \mathrm{P\_Env\_PS\_BRK\_ERR} = \mathbf{T} \vee \\
\quad (\mathrm{P\_Env\_PS\_CLT\_PRSD} = \mathbf{T} \wedge \mathrm{P\_Env\_PS\_BRK\_ERR} = \mathbf{F} \wedge \\
\quad \mathrm{P\_Env\_PS\_CLT\_ERR} = \mathbf{F} \wedge \mathrm{P\_Env\_PS\_ACC\_ERR} = \mathbf{F}) \vee \\
\quad \mathrm{P\_Env\_PS\_CLT\_ERR} = \mathbf{T} \ \vee \ \mathrm{P\_Env\_PS\_ACC\_ERR} = \mathbf{T}
\end{array}
$$

The (DLF) proof obligation in general quickly becomes very complex, in particular, if the involved events have parameters. As long as it holds and is discharged by an automatic theorem prover this is not a problem. The more common situation is, however, that it cannot be proved. As a matter of fact, this is also the more interesting situation because it may point to errors in the model. For example, the (DLF) proof obligation above could not be proven: an event for covering the case in which all pedal signals are set to FALSE was missing.

We need the tool in order to find errors in our model and expect support for correcting the model. To put this into perspective: the real model of the case study from which the example has been extracted has 78 constants with 121 axioms, 62 variables with 59 invariants and has 80 events with 855 guards. When the proof of this proof obligation fails, it is not clear at all why this is. Analysis of a large failed (DLF) proof obligation by interactive proof is very time consuming. Maybe we simply do not find the proper proof; maybe there is a deadlock; maybe the invariant is too weak. Counter examples can provide vital clues where to look for problems.

A different approach is needed to check for deadlocks. The most immediate is to animate the model and see whether we encounter a state in which no event is enabled. Another one is to model-check it. Model checking can provide fast feedback, but is also associated with known problems: in many applications the state space is either infinite or much too large to explore exhaustively. In the case study model checking did not produce satisfactory results. Neither proof nor model-checking worked.

Constraint checking. Finally, we did achieve good results using constraint checking. We have implemented a dedicated constraint checker to deal with deadlock-freedom [5]. It is not based on model execution but yields a solution to the (DLF) proof obligation, providing a counter example if (DLF) does not hold. On sub-models with about 20 events constraint checking was very effective in helping to develop a correct deadlock-free model. But on the large proof obligation mentioned above it did not help to resolve all problems. Constraint solving computed counterexamples to (DLF) but eventually it became too time consuming to see how the model could be corrected. No obvious improvement to

[4] Currently, this proof obligation is not generated by the Rodin tool. We have generated it by means of the Flow-Plugin [7].

ProB could have solved this problem. Independently of the modelling method, the large number of constants and variables makes it difficult to interpret deadlocked states and understand how the model has to be corrected. We believe that refinement can be used to address the inherent complexity of the model. This way deadlock-freedom could be analysed for models whose size is increased in small increments: we have already seen that dealing with about 20 events at once is effectively possible.

Lessons learned. The lesson we learned is that the problem could only be solved by using multiple verification techniques, such as proof, model checking and animation, in order to analyse, understand and debug the formal model. Matching Problem Frames with Event-B may have to be relaxed allowing for refinement in Event-B to cater for a stepwise introduction of records. The most interesting insight we gained from the case study was the need for strong tools to allow for large models and at the same time the need for appropriate techniques to reduce the size of those models. We plan to redo some parts of the modelling to find a good measure for that mix. The case study was important as a driver for tool improvements, in particular, of ProB. The scale of the case study was the key to this.

6 Conclusion

When we started the case study we asked whether Event-B is fit for industrial use. We had to be more specific about our question. There are too many factors besides Event-B so that we should have asked whether Event-B fits into a suitable methodology for development (at Bosch). We have seen that Event-B allowed us to think about properties of the cruise system model that would be difficult to achieve non-formally. We have matched Problem Frames elaboration formally using Event-B refinement. The resulting method of Event-B elaboration supported by theory and tools is novel and was crucial for the use of Event-B in the targeted development process. We have successfully analysed the model with respect to deadlock-freedom, but saw the difficulty of using the counter examples to develop a non-trivial deadlock-free system. We believe that refinement will be the key to overcome this difficulty: the model must be analysed and constructed piecemeal to provide better feedback to the engineers. In future work, we would like to check more properties. For instance, check wether the choice between the events of a machine is deterministic. Only one event should be enabled at any time: we expect an implementation of a cruise control controller to be predictable. We also would like to analyse certain sequences of actions using temporal (LTL) formulas. Some requirements do not fit into the simple scheme of events and invariants.

What about the answer to our question? In the case study we could see that we can clearly profit from the use of formal methods in the development process. Checking the model for consistency and deadlock-freedom uncovered many errors in the model and led to various improvements of the requirements. For example, deadlock checking identified many cases in which requirements had been missing.

Event-B with its tools Rodin and ProB can be useful for improving the quality of requirements and of models that can serve as blue prints for implementations. We have also seen that we have profited in "unintended" ways. We did not follow the method described in [1] but had to develop our own, in particular, to work with Problem Frames for the management of complex requirements.

Acknowledgements. We are especially grateful to Cliff Jones who coordinated the work of the academic partners in this case study. We are also grateful for the fruitful interactions with Rezazadeh Abdolbaghi, Jean-Raymond Abrial, Michael Butler, Alexei Iliasov, Michael Jackson, Sascha Romanovsky, Matthias Schmalz, and Colin Snook.

References

1. J.-R. Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010.
2. Jean-Raymond Abrial, Michael J. Butler, Stefan Hallerstede, Thai Son Hoang, Farhad Mehta, and Laurent Voisin. Rodin: an open toolset for modelling and reasoning in Event-B. *STTT*, 12(6):447–466, 2010.
3. Andrew Edmunds and Michael Butler. Tool support for Event-B code generation, 2009.
4. Neil Evans and Michael J. Butler. A proposal for records in Event-B. In Jayadev Misra, Tobias Nipkow, and Emil Sekerinski, editors, *FM*, volume 4085 of *Lecture Notes in Computer Science*, pages 221–235. Springer, 2006.
5. Stefan Hallerstede and Michael Leuschel. Constraint-Based Deadlock Checking of High-Level Specifications. *Proceedings ICLP'2011*, pages –, 2011.
6. Stefan Hallerstede, Michael Leuschel, and Daniel Plagge. Refinement-animation for Event-B - towards a method of validation. In Marc Frappier, Uwe Glässer, Sarfraz Khurshid, Régine Laleau, and Steve Reeves, editors, *ABZ*, volume 5977 of *Lecture Notes in Computer Science*, pages 287–301. Springer, 2010.
7. Alexei Iliasov. On Event-B and Control Flow. Technical Report CS-TR-1159, University of Newcastle, 2009.
8. Michael Jackson. *Problem Frames: Analyzing and structuring software development problems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.
9. Cliff B. Jones. DEPLOY Deliverable D15: Advances in Methodological WPs. http://www.deploy-project.eu/pdf/D15final.pdf.
10. Cliff B. Jones. *Systematic Software Development Using VDM*. Prentice Hall, second edition, 1990.
11. Michael Leuschel and Michael J. Butler. ProB: an automated analysis toolset for the B method. *STTT*, 10(2):185–203, 2008.
12. Michael Leuschel, Jérôme Falampin, Fabian Fritz, and Daniel Plagge. Automated property verification for large scale B models. In A. Cavalcanti and D. Dams, editors, *Proceedings FM 2009*, volume 5850 of *Lecture Notes in Computer Science*, pages 708–723. Springer, 2009.
13. Felix Loesch, Rainer Gmehlich, Katrin Grau, Cliff B. Jones, and Manuel Mazzara. DEPLOY Deliverable D19: Pilot Deployment in the Automotive Sector. http://www.deploy-project.eu/pdf/D19final.pdf.
14. Colin F. Snook and Michael J. Butler. UML-B: Formal modeling and design aided by UML. *ACM Trans. Softw. Eng. Methodol*, 15(1):92–122, 2006.

15. Olaf Stursberg, Ansgar Fehnker, Zhi Han, and Bruce H. Krogh. Verification of a cruise control system using counterexample-guided search. *Control Engineering Practice*, 12(10):1269 – 1278, 2004. Analysis and Design of Hybrid Systems.
16. Sanaz Yeganefard, Michael Butler, and Abdolbaghi Rezazadeh. Evaluation of a Guideline by Formal Modelling of Cruise Control System in Event-B. In César Muñoz, editor, *Proceedings of the Second NASA Formal Methods Symposium (NFM 2010)*, number NASA/CP-2010-216215, NASA Langley Research Center, Hampton VA 23681-2199, USA, April 2010.